

EXHIBIT B



US009348622B2

(12) **United States Patent**
Emelyanov et al.

(10) **Patent No.:** **US 9,348,622 B2**
(45) **Date of Patent:** **May 24, 2016**

(54) **METHOD FOR TARGETED RESOURCE
VIRTUALIZATION IN CONTAINERS**

(58) **Field of Classification Search**

None

See application file for complete search history.

(71) Applicants: **Pavel Emelyanov**, Moscow (RU); **Igor Petrov**, Moscow (RU); **Stanislav S. Protassov**, Moscow (RU); **Serguei M. Beloussov**, Singapore (SG)

(56) **References Cited**

U.S. PATENT DOCUMENTS

7,349,970 B2 * 3/2008 Clement G06F 9/5033
709/201
2005/0091652 A1 * 4/2005 Ross G06F 9/45533
718/1
2013/0139157 A1 * 5/2013 Koh G06F 9/4555
718/1

* cited by examiner

(73) Assignee: **Parallels IP Holdings GmbH**,
Schaffhausen (CH)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 179 days.

Primary Examiner — Gregory A Kessler

(74) *Attorney, Agent, or Firm* — Bardmesser Law Group

(21) Appl. No.: **14/251,989**

(22) Filed: **Apr. 14, 2014**

(65) **Prior Publication Data**

US 2015/0150003 A1 May 28, 2015

(30) **Foreign Application Priority Data**

Nov. 26, 2013 (EA) 201301283

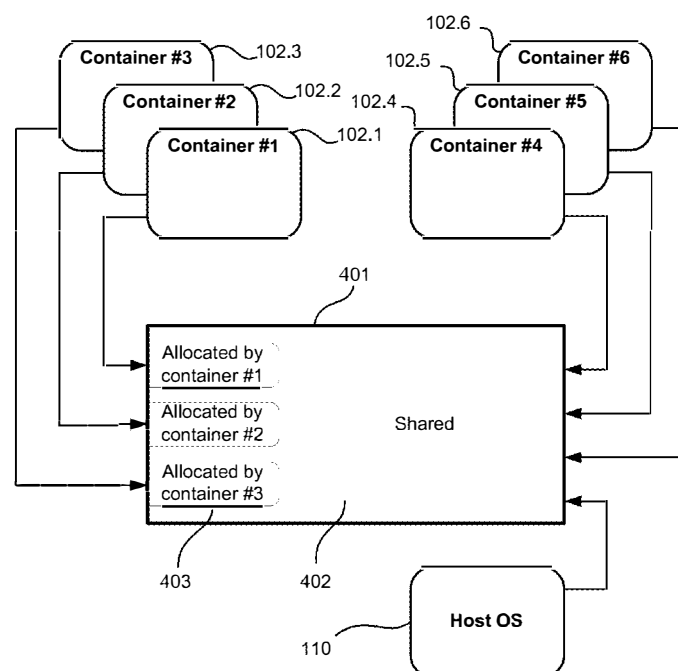
(51) **Int. Cl.**
G06F 9/455 (2006.01)

(52) **U.S. Cl.**
CPC **G06F 9/455** (2013.01); **G06F 9/4555**
(2013.01)

(57) **ABSTRACT**

A method and computer program product for targeted container virtualization, where only separate components of a computer system or a server are virtualized. The OS kernel and other server resources are not virtualized. Only selected components—applications or resources are targeted for virtualization instead of virtualization of the entire system. Targeted virtualization provides for more flexible container isolation from each other and from a host node. This, in turn, provides for optimized more flexible cloud infrastructure. Each element within a container virtualization model is optional in terms of virtualization. The element's virtualization option can be turned on and off by an administrator or by a client who owns the container.

17 Claims, 9 Drawing Sheets



Conventional Art

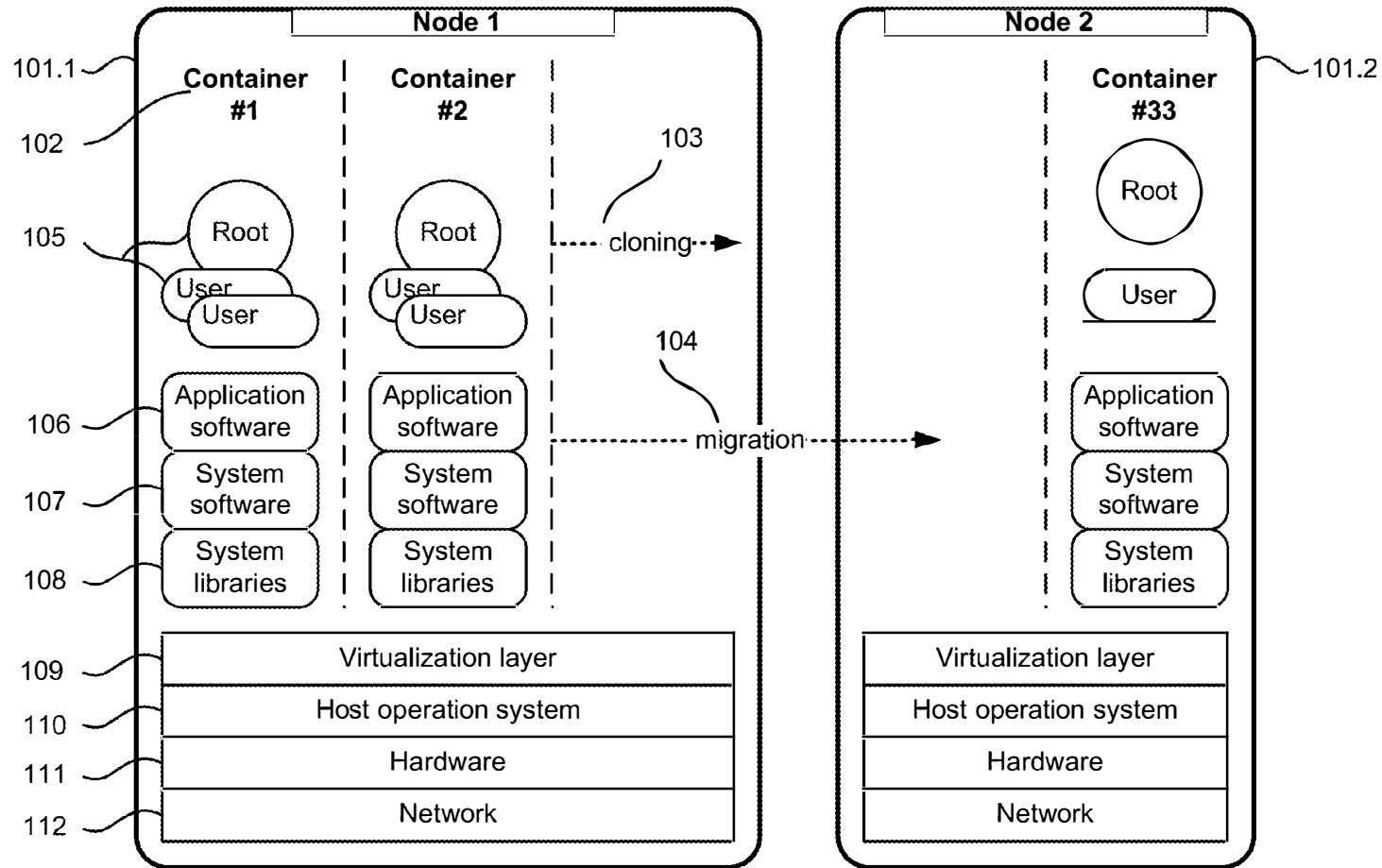
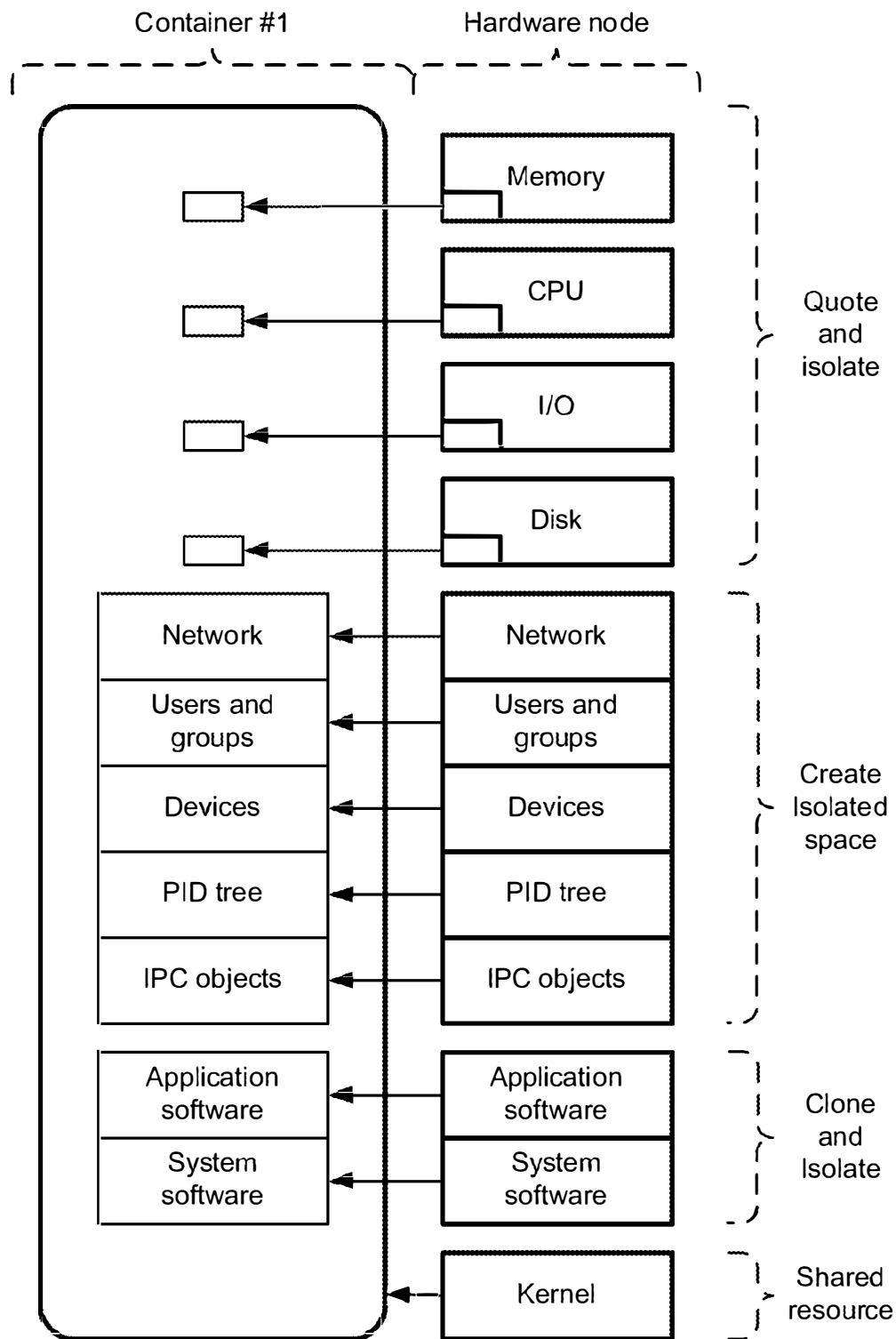


FIG. 1

Conventional Art**FIG. 2**

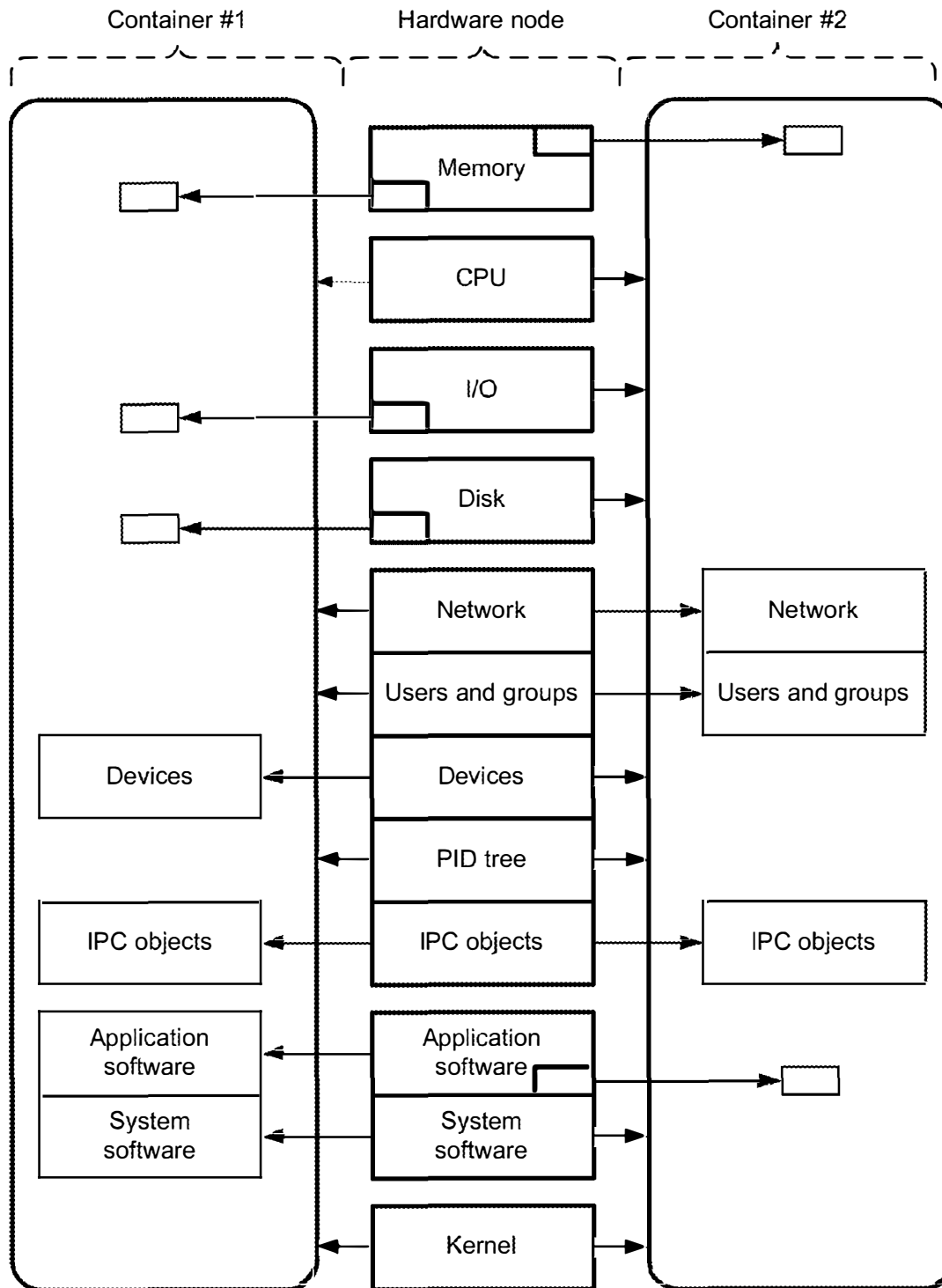


FIG. 3

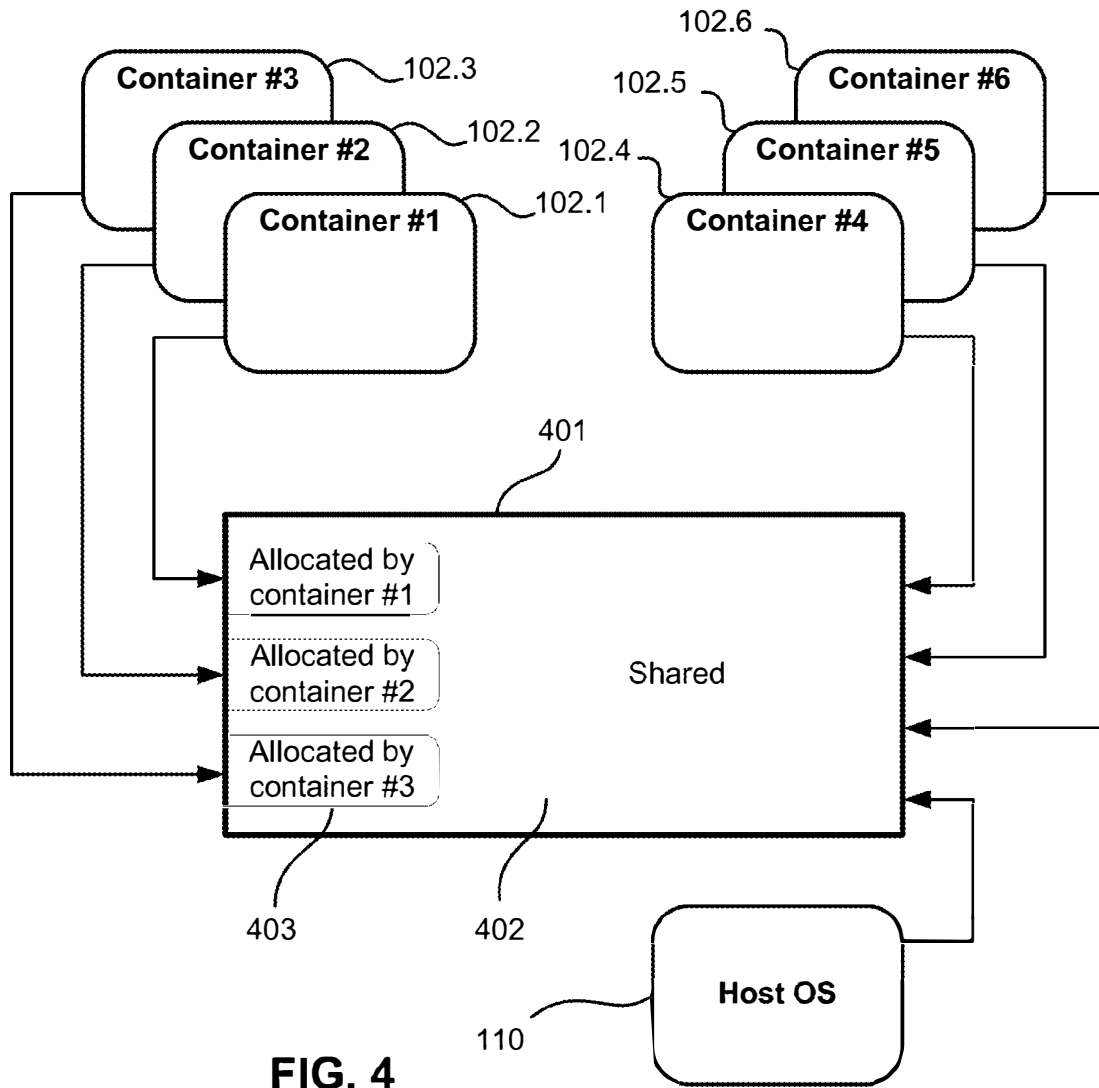


FIG. 4

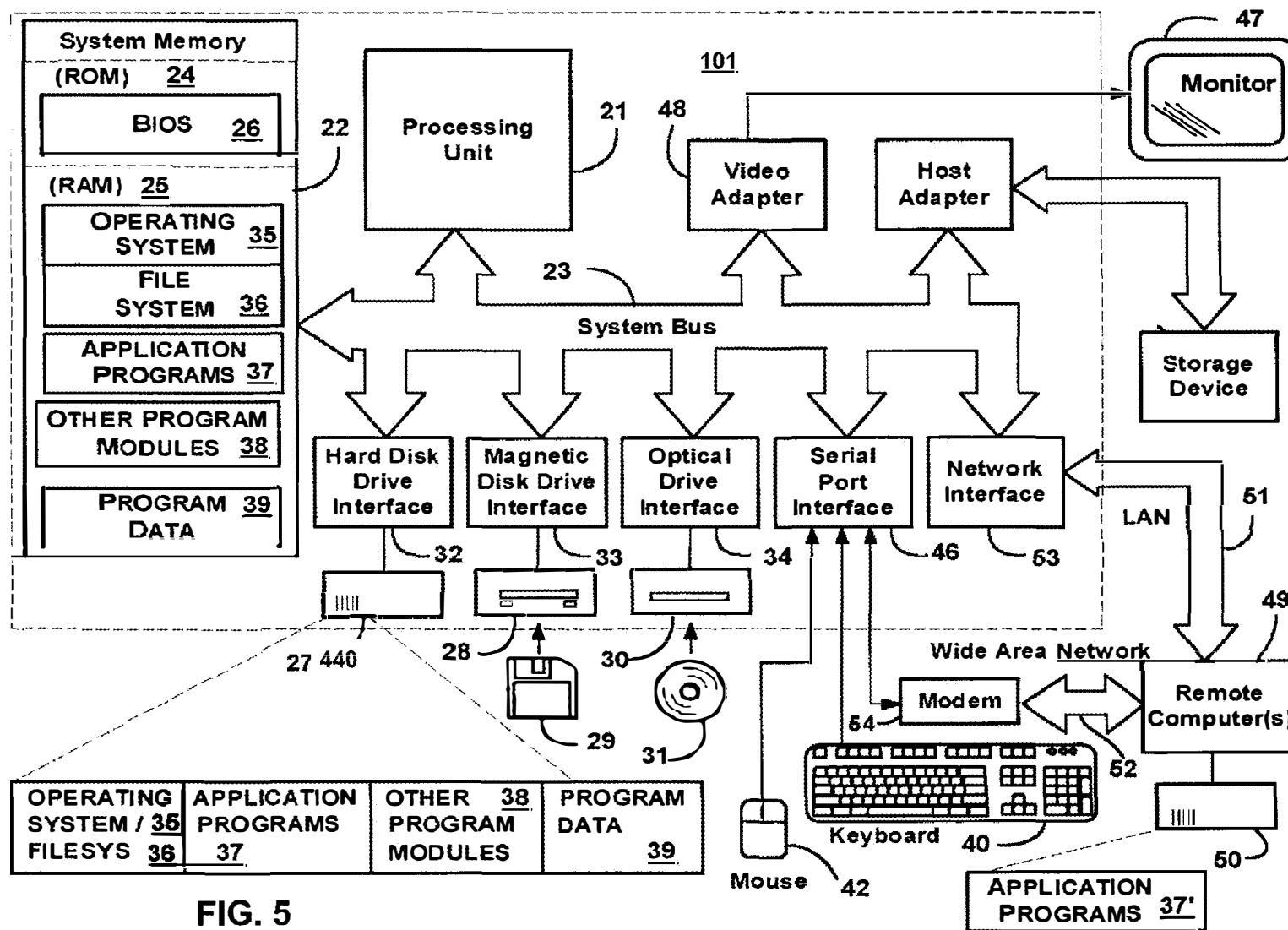


FIG. 5

U.S. Patent

May 24, 2016

Sheet 6 of 9

US 9,348,622 B2

```

-$ cat /proc/user_beancounters
Version: 2.5

```

uid	resource	held	maxheld	barrier	limit	failures
221:	kmemsize	1048088	10823454	9223372036854775807	9223372036854775807	0
	lockedpages	0	0	9223372036854775807	9223372036854775807	0
	privpages	134181	134222	9223372036854775807	9223372036854775807	0
	shpages	671	671	9223372036854775807	9223372036854775807	0
	dummy	0	0	9223372036854775807	9223372036854775807	0
	numproc	37	37	9223372036854775807	9223372036854775807	0
	physpages	42367	42366	9223372036854775807	9223372036854775807	0
	vmmappages	0	0	9223372036854775807	9223372036854775807	0
	comparpages	42367	42366	9223372036854775807	9223372036854775807	0
	numtapesack	7	7	9223372036854775807	9223372036854775807	0
	numlock	8	8	9223372036854775807	9223372036854775807	0
	numpty	1	1	9223372036854775807	9223372036854775807	0
	numsiginfo	0	1	9223372036854775807	9223372036854775807	0
	topendbuf	124836	124836	9223372036854775807	9223372036854775807	0
	toporvbuf	114800	114800	9223372036854775807	9223372036854775807	0
	othersockbuf	1160	11640	9223372036854775807	9223372036854775807	0
	dgramorvbuf	0	0	9223372036854775807	9223372036854775807	0
	numothersock	13	14	9223372036854775807	9223372036854775807	0
	docheatle	2160839	2160836	9223372036854775807	9223372036854775807	0
	numfile	3043	3044	9223372036854775807	9223372036854775807	0
	dummy	0	0	9223372036854775807	9223372036854775807	0
	dummy	0	0	9223372036854775807	9223372036854775807	0
	dummy	0	0	9223372036854775807	9223372036854775807	0
	numiprcnt	10	10	9223372036854775807	9223372036854775807	0

FIG. 6

Conventional Art

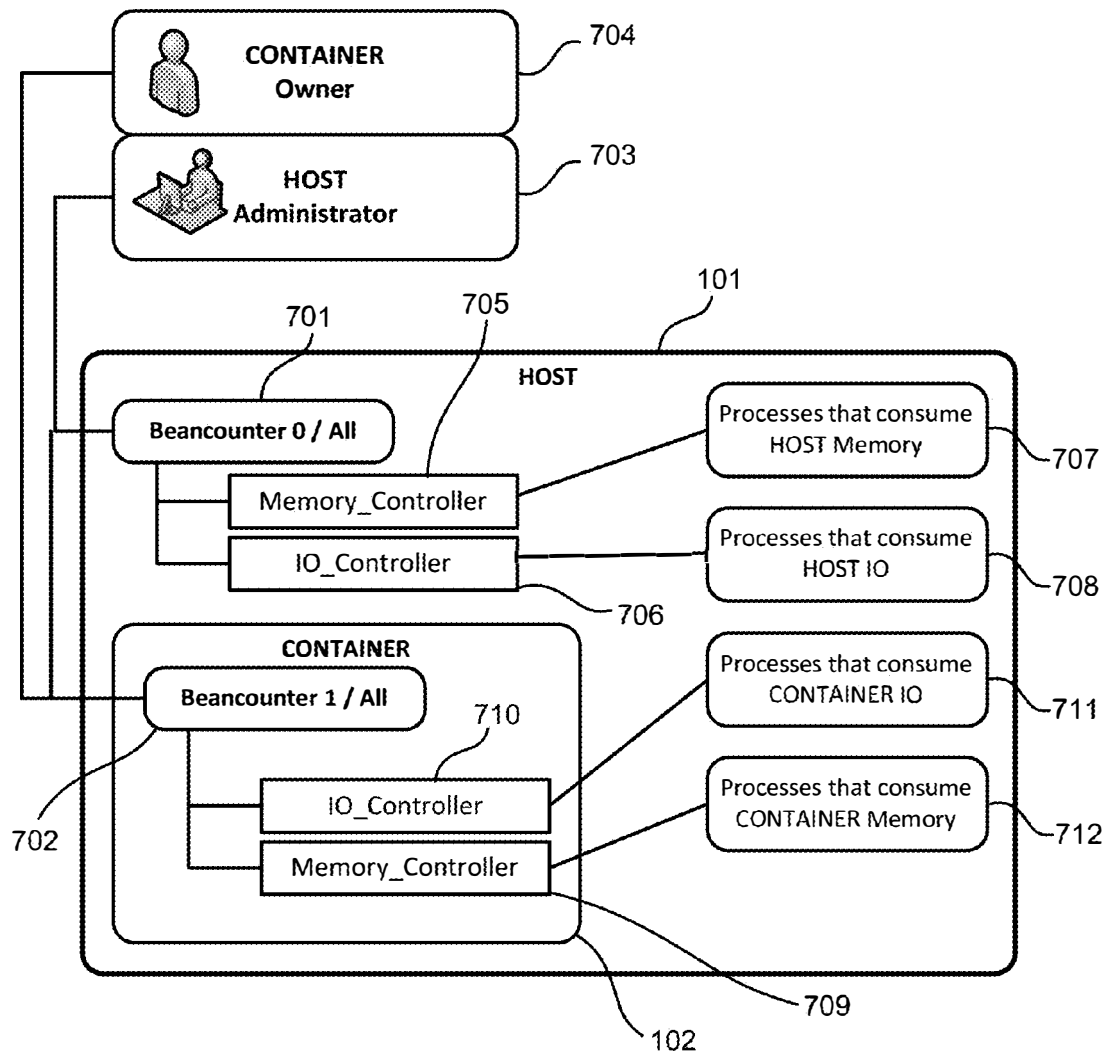


FIG. 7

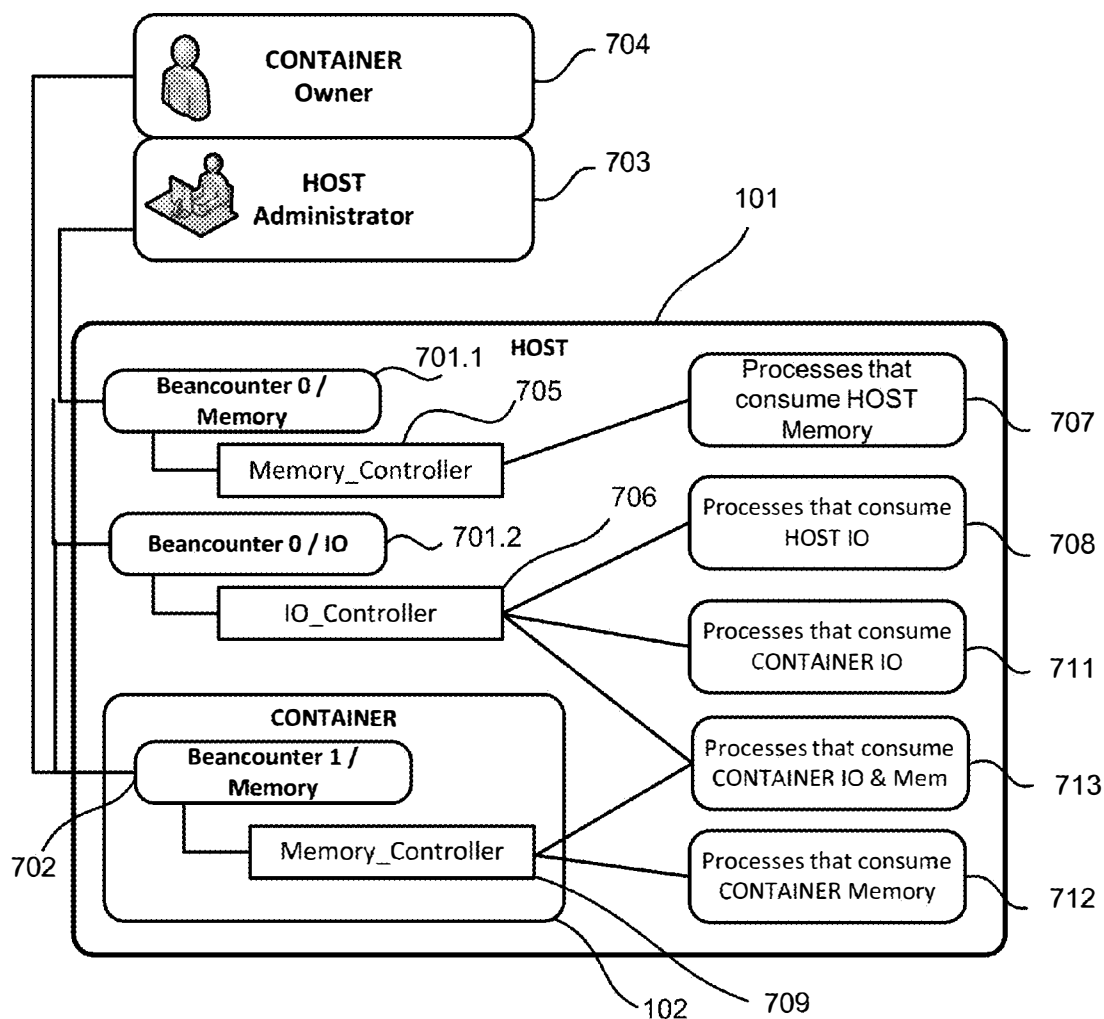


FIG. 8

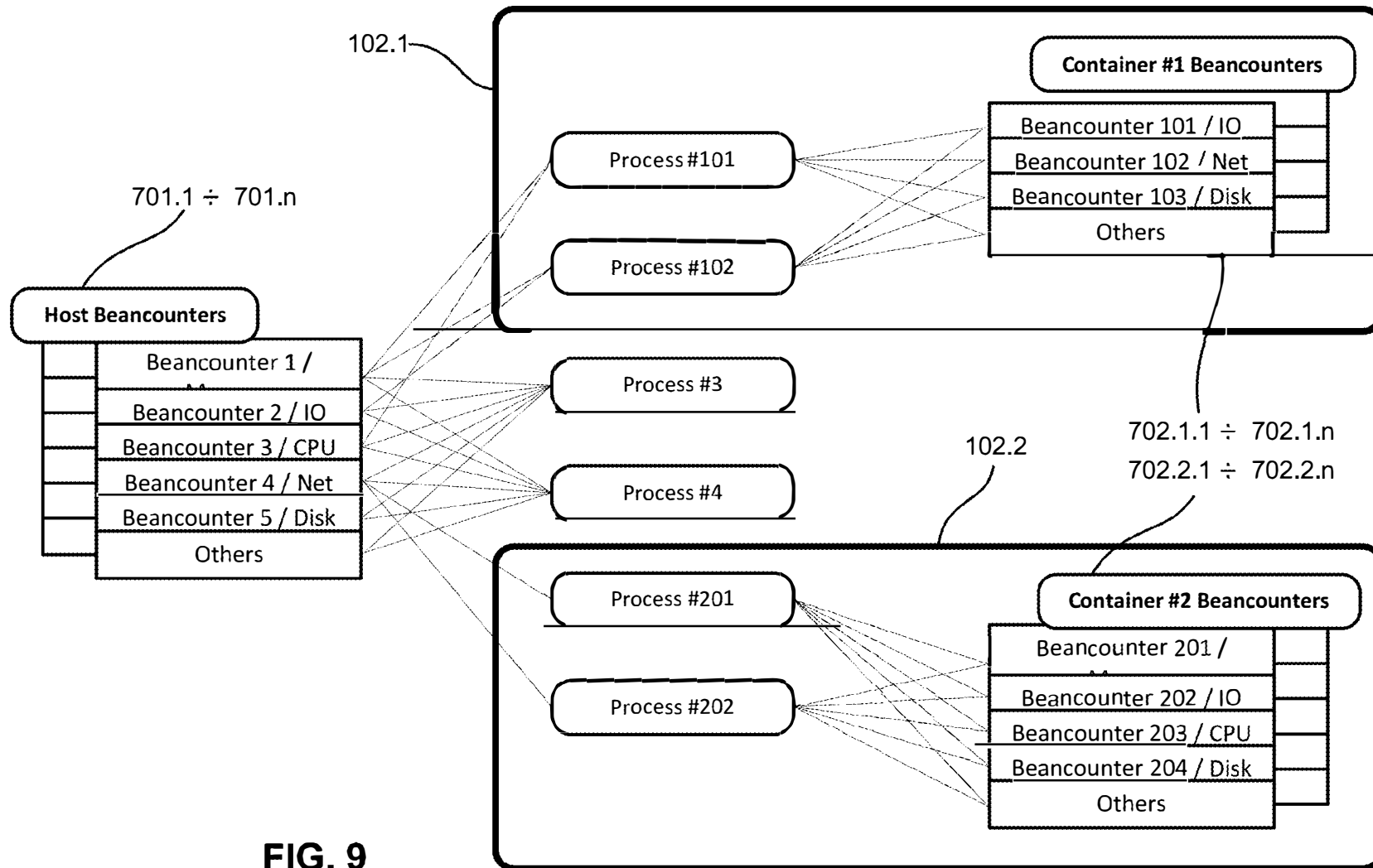


FIG. 9

US 9,348,622 B2

1

METHOD FOR TARGETED RESOURCE VIRTUALIZATION IN CONTAINERS

CROSS-REFERENCE TO RELATED APPLICATIONS

This application claims priority to Eurasian Application No. 201301283, filed on Nov. 26, 2013, which is incorporated herein by reference in its entirety.

BACKGROUND OF THE INVENTION

1. Field of the Invention

The present invention relates to a method for targeted container virtualization where only separate components of a computer system or a server are virtualized.

2. Description of the Related Art

Typically, virtualization at an OS kernel level provides for several isolated user spaces—containers. The containers are identical to a real server from a user point of view. Each container has entire OS components except for the kernel. Thus, the applications from different containers cannot interact with each other.

The containers have allocated resource quotes. For example, a hardware node can have thousands of containers running on it. Each of these containers can only use allocated by quota amount of every server resource. The typical resources are: disk space, I/O operations, operating memory, CPU time, network traffic, etc. The containers have their own sets of user groups, processes and devices isolated from the host OS and from other containers. Virtualization provides for isolation and limitation of resources in the containers.

The containers can be migrated to other host servers almost without any interruptions of their services by a “live migration.” The conventional OS-level virtualization solutions are, e.g.: FreeVPS, Icore virtual accounts, FreeBSD Jails, Linux-VServer, OpenVZ, Virtuozzo, Solaris Zones.

Linux kernel with the OpenVZ uses two entities: cgroups (control groups) and namespace. Both of these entities are needed for limiting and controlling the resources allocated for a particular process. For example, a namespace of a network subsystem is a standard network subsystem, a network configuration stack and iptables. The namespaces are created for a memory, for a processor, for a disk and for all other Linux kernel subsystems. Another namespace can be created on the same host system and the two instances can be used separately. For example, in order to allow a container application to apply its own parameters for working with the network. The application can have large buffers, use sockets and use proprietary configuration setting for iptables.

CGroups is a set of limitations for a set of kernel processes. For example, Apache, PHP, MySQL can be encapsulated into a cgroup. Then, the following limitations can be set: CGroup with a modifier webstack can use not more than 10 GB on aggregate, can use not more than 5 processes per CPU Units, can use not more than 3 MB for site cache, etc. Cgroups allow for allocation of resources, such as CPU time, system memory, network bandwidth, or combinations of these resources among user-defined groups of tasks (processes) running on a system. The cgroups can be monitored and configured. The cgroups can be denied access to certain resources. The cgroups can be reconfigured dynamically on a running system. The cgconfig (control group config) service can be configured to start up at boot time and reestablish the predefined cgroups, thus making them persistent across reboots. By using cgroups, system administrators gain fine-grained control over allocating, prioritizing, denying, man-

2

aging, and monitoring system resources. Hardware resources can be efficiently divided up among the tasks and the users, increasing overall efficiency.

Virtual Machines (VMs) use hypervisor-based virtualization, where the OS kernel is virtualized for each of the VMs in a form of a Guest OS. Container virtualization uses one instance of an OS kernel of the physical host for all of the containers. Therefore, the containers usually work faster than the VMs, and the density of the containers implemented on one physical machine can be much higher than that of the VMs.

Thus, the containers are widely used by hosting providers. A hoster creates a separate container for each of its clients. These containers are, from the user’s perspective, practically identical to a physical server in terms of their functionality. Such containers are also sometimes referred to as Virtual Private Servers (VPS). Typically, the containers are employed by service providers using a cloud-based infrastructure. Complete virtualization of all of the cloud resources is often not needed and can be costly. However, none of the existing systems offer a selected or targeted virtualization of separate cloud components or resources.

For example, a clustered system for storing, searching and processing images may only require launching a couple of processes in a virtual environment for editing client’s pictures. In this case, virtualization of a network is not required. Virtualization of a user space and creation of process identifiers (pid) is also not needed. Only the operating memory needs to be virtualized.

Another example requiring targeted (selected) virtualization is a backup service. The backup is limited by a disk capacity, because it takes a large amount of data from one disk and places it onto another disk. Therefore, only the disk space and I/O resources need to be virtualized.

Another example that requires only partial (targeted) virtualization is APACHE web server. The web server provides shared hosting—a number of directories containing web-sites. If the web server needs to open a site upon the http-request, the web-server launches a special process, which serves the http-request by taking data from a certain directory. This process needs to be launched in and isolated environment only using some limited resources. In other words, the entire Linux OS does not need to be virtualized. Only the disk space, I/O, memory and network need to be virtualized. Thus, targeted virtualization (or fragmented isolation) is more efficient than the conventional virtualization such as a conventional system depicted in FIG. 1.

FIG. 1 illustrates conventional container virtualization. The hardware nodes 101.1 and 101.2 can have a plurality of containers 102 implemented on them. Each of these containers has some resources isolated from the host OS 110 and from the other containers. These resources are users and groups 105, system utilities 107, applications 106 and other resources. The resource allocation, isolation and control of the containers are supported by the virtualization layer 109 implemented on the host OS kernel. The host node has hardware components 111 and a network 112. The container 102 can be cloned (103) by creating a copy of the container with a different identifier. Migration 104 of the container 102 includes moving all container files (applications 106, system software 107 and system libraries 108) and its configuration to another node.

FIG. 2 illustrates a conventional schema of container virtualization in more detail. Note that not all (out of about 20) possible virtualization parameters are illustrated. FIG. 2 illustrates different conventional approaches used for isolation of the resources. For example, CPU time is allocated to the

US 9,348,622 B2

3

container by the hardware node using a special quota. A conventional process planner has two levels. At the first level, the planner decides to which process to give a quant of the CPU time based on a pre-set quota for this container. At the second level, the process planner decides to which process in the given container to give the quant of the CPU time based on standard priorities of the OS kernel (e.g., Linux system).

Setting resource usage limits in the containers is flexible. Therefore, the containers are used in hosting, in developing, testing and running corporate sites. For example, several containers with different configuration can be created on a host and each of the containers can have allocated memory so that the aggregated memory for all of the containers exceeds the memory of the host. Even one container can have allocated memory that exceeds the host memory. Then, if the memory is needed, the memory swap (transfer memory content to the disk) can be used. If the swap is completed, but the memory is still needed, the utility OOM Killer of the host is launched. This utility “kills” the most memory consuming processes in the containers. Thus, the container memory can be controlled in a more flexible manner than a memory of the VMs. When an aggregate memory of the containers exceeds the host memory, this is very convenient for developing tasks.

Likewise, the resources such as disk space, I/O are also allocated to the container according to the corresponding quotas. Some other resources, for example, a network, users and groups, devices, PID tree, IPC objects are virtualized on the container within an isolated space. Other components, such as applications, system software are cloned to on the container. The kernel resources are made available to a container or to multiple containers. In this conventional schema all of the resources are virtualized in the container regardless of the container needs. The container may not need all of the resources as shown in the above examples. Thus, a system, where only resources required by the container are virtualized, is desired.

Accordingly, there is a need in the art for a method for targeted or selected virtualization of components of the containers. Additionally, there is a need in the art for a flexible cloud infrastructure based on more flexible container isolation.

SUMMARY OF THE INVENTION

Accordingly, the present invention is directed to a method and system for targeted container virtualization, where only separate components of a computer system or a server are virtualized that substantially obviates one or more of the disadvantages of the related art.

The OS kernel is not virtualized. Other server resources can be optionally virtualized. In other words, only selected components—applications or resources are targeted for virtualization instead of the virtualization of the entire system. Targeted resource virtualization provides for more flexible container isolation from each other and from a host node. This, in turn, provides for optimized more flexible cloud infrastructure.

According to an exemplary embodiment, each element within a container virtualization model is optional in terms of virtualization. The element virtualization option can be turned on and off by an administrator or by a client who owns the container. Such an Application Container does not launch the entire set of OS components. Instead, it launches only some required container processes. Note that these processes are limited according to what the client needs and not by hard quotas.

4

An innovative concept of the Application Containers is based on OpenVZ solution. If host wants to see list of all running processes, the ps-lc command is issued. This command shows all host processes including the processes running inside the container. The command find (for finding the files) shows all files including the files inside the container. This is possible because the host and the container operate in the same address, disk and network space. Thus, the host has additional means for controlling the containers.

Additional features and advantages of the invention will be set forth in the description that follows, and in part will be apparent from the description, or may be learned by practice of the invention. The advantages of the invention will be realized and attained by the structure particularly pointed out in the written description and claims hereof as well as the appended drawings.

It is to be understood that both the foregoing general description and the following detailed description are exemplary and explanatory and are intended to provide further explanation of the invention as claimed.

BRIEF DESCRIPTION OF THE ATTACHED FIGURES

The accompanying drawings, which are included to provide a further understanding of the invention and are incorporated in and constitute a part of this specification, illustrate embodiments of the invention and together with the description serve to explain the principles of the invention.

In the drawings:

FIG. 1 illustrates a conventional containers located on different host nodes;

FIG. 2 illustrates a conventional use of resource by a container;

FIG. 3 illustrates a targeted virtualization schema, in accordance with the exemplary embodiment;

FIG. 4 illustrates application container virtualization for memory, in accordance with the exemplary embodiment;

FIG. 5 illustrates a schematic diagram of an exemplary computer or server that can be used in the invention;

FIG. 6 illustrates the output of the command `<<cat/proc/user_beancounters>>` in accordance with the exemplary embodiment;

FIG. 7 illustrates the use of Beancounters in a standard container virtualization schema;

FIG. 8 and FIG. 9 illustrate the use of Beancounters in container virtualization in accordance with the exemplary embodiment.

DETAILED DESCRIPTION OF EMBODIMENTS OF THE INVENTION

Reference will now be made in detail to the embodiments of the present invention, examples of which are illustrated in the accompanying drawings.

The present invention is directed to a method for targeted container virtualization, where only separate components of a computer system or a server are virtualized that substantially obviates one or more of the disadvantages of the related art.

The OS kernel and some server resources are not virtualized. Only selected components—applications or resources are targeted for virtualization instead of virtualization of the entire system. Targeted virtualization provides for more flexible container isolation from each other and from a host node. This, in turn, provides for optimized more flexible cloud infrastructure, including multiple containers.

US 9,348,622 B2

5

According to an exemplary embodiment, each element within a container virtualization model is optional in terms of virtualization. The element virtualization option can be turned on and off by an administrator or by a client who owns the container. Such Application Container does not launch the entire set of the OS components. Instead, it launches only some required container processes. Then, these processes are limited according to what the client needs and not by hard quotas.

Examples of virtualized resources can be, e.g.:

Memory—i.e., hardware shared memory, number of RAM pages;

I/O operations—in computing, input/output or I/O is the communication between an information processing system (such as a computer) and the outside world, possibly a human or another information processing system. Inputs are the signals or data received by the system, and outputs are the signals or data sent from it.

Disk—number of open files, amount of swap space.

Network—isolated stack, e.g., links, addresses, routes, netfilter rules and sockets.

Users and groups—if it is virtualized, then users in container have internal accounting (user with uid=3, 4). The host has its own and separate users with uid=3, 4. If it is not virtualized container has uid=3, 4 and host has uid=5, 6 . . . (rather than uid 3 and 4).

Devices—either per-container access rights to devices (e.g., shared disk can only be read or mounted with file system by containers), or different mappings from device ID (major: minor pair in Linux) into real hardware.

PID tree—similar to the situation with users described above.

IPC objects—if virtualization is turned on, then container and host have separated queues of requests to the processor.

User applications—for example, there is an application “apt-get” for software updating. If virtualization is turned on, then apt-get in the container and in the host are different applications in different folders and have files with a different config. If virtualization is off, then apt-get is a single application on the host that can be used by container as well.

System modules—similar to the situation with applications.

FIG. 3 illustrates a targeted virtualization schema, in accordance with the exemplary embodiment. FIG. 3 illustrates Application Container virtualization for two containers. The Container #1 and the Container #2 can require different level of virtualization. In other words, the Containers require a different set of virtualized resources. For example, the Container #1 needs virtualization of: Memory, I/O, Disk, Devices, IPC objects, Applications and System software. The Container #1 uses these resources in isolation and/or by allocation quota. The Container #1 uses the following resources CPU, Network, Users and groups, PID tree as shared resources with the OS kernel. The OS kernel may not be virtualized and is used in the same manner as in the conventional virtualization scheme. However, the OS kernel includes some container virtualization components. These virtualization components can be turned if required by a particular application container.

The exemplary Container #2 uses only virtualization of memory, Network, Users and groups, IPC objects and Application software. Container #2 uses CPU, I/O, Disk, Devices, PID tree and System software as shared resources from the host node that are not virtualized. Thus, targeted virtualization is implemented for the Container #1 and the Container #2. The illustrated targeted virtualization approach advantageously saves the disk space. The targeted virtualization approach allows for a larger number of containers that can be

6

implemented on a single hardware node. The proposed approach provides for more flexibility in managing the containers. It also provides for higher efficiency if I/O or other resources are not virtualized. In general, any resource that is not virtualized improves efficiency of the system.

A user of the container can, advantageously, turn on and off the resources based on a current need. Components of virtualization can be turned ON/OFF by means of, for example: a) command line with needed parameters; b) commenting on/off of lines with references to resource being virtualized in a configuration file; c) GUI software, if it is exists. This provides a user with more flexibility and makes containers more suitable for use in a cloud infrastructure. FIG. 4 illustrates Application container virtualization of memory by several containers. The containers 102.1, 102.2 and 102.3 use virtualized memory. The containers 102.1, 102.2 and 102.3 have allocated (according to a quota) isolated memory sections 403 within a shared memory 401. However, the containers 102.4, 102.5 and 102.6 do not use virtualized memory. Instead, these containers use a shared memory area 402 of the memory 401 together with some of the containers implemented on the host OS 110.

According to the exemplary embodiment, the container application can see that the container is divided into several parts. Some parts can be completely isolated from other containers and from the host OS. Some other parts can be used together with the host OS and other containers. In case of Linux, the OS kernel is modified. A set of special utilities is provided for creation and management of the containers.

Conventional systems, upon creation of the container, set up resource usage quotas for each of the resources (or they are set by default). According to the exemplary embodiment, if a quota is not set for a particular resource, the resource is shared by all of the containers that do not have this resource virtualized and allocated. Note that the exemplary embodiment can be implemented with any OS.

According to the exemplary embodiment, a container creator (a person or an application) can see an on/off switch for each of the container resources. For example, if “Virtualize Memory” switch is on, the memory is virtualized for the given container. This means that the memory for this container is allocated according to a certain quota and nobody else can use the allocated memory. If the “Virtualize Memory” switch is off, the container uses the shared memory of the hardware node. Additionally, the hoster can share some resources between containers. For example, a user can have a hosted site, which has a low number of requests. The hoster provides statistics to the site owner regarding consumption of resources and quotas, such as memory, I/O, processor, disk, database, and so on. The statistics reflect actual usage of the resources. As the number of visitors to the site grows, the frequency of exceeding the quotas also grows, which can trigger “abuse notices” from the hoster. However, many hosters would not disable the site, despite it going over the quota—in a sense, this is a consequence of sharing resources with other users, whose sites have lower resource usage than their quotas permit. Conventional co-location or Virtual Private Servers would not permit this, since the quotas are “set in stone.” On the other hand, an experienced user can configure his environment in an optimal manner.

The client can see the switches in his hosting control panel, and the administrator can see the switches in a configuration text file. For example, if the line with memory is commented, then virtualization is switched off (for shared memory), otherwise, if there is no comment, then memory virtualization is switched on (the user cannot “take” more memory than his quota, and will pay only for what the quota allows).

US 9,348,622 B2

7

According to the exemplary embodiment, the configuration file can have different formats. However, the resource can be only in one of the states: on or off (1 or 0). An example of the configuration file is:

```
100.conf:
memory_manager=on
io_manager=on
network_virt=on
pid_virt=off
```

This means that the container with the id=100 has virtualization turned on for memory, IO and network resources. Virtualization of the pid is turned off.

If a command line “vzctl start 100” is executed, then the container with the id=100 uses all values from the 100.conf file without any modifications. The command line “vzctl start 100 -io_manager=off” means that all resources are taken from the configuration file, but the io-manager state is taken from the command line.

According to the exemplary embodiment, the container isolation rules are made somewhat less restrictive and more flexible. Each container has its own configuration file indicating which virtualization components should be used. Note that the containers have the same sets of resources, but different containers can have different virtualization rules for IO, CPU, PIDs, network, etc. This allows for a more flexible and more efficient cloud infrastructure. In the cloud we can define three types of the containers:

1. A resource-only container, which has only memory, CPU, IO and net bandwidth resource limited and controlled. This is useful for cloud services, that are not supposed to be migrated from one node to another;
2. A resource+namespaces container, which is type 1+virtualized IDs, like PIDs, net stack, IPC, etc. This container type can be good for the same use-case as in type 1, but with an ability to migrate it to another host for load balancing; and
3. Type 2+isolated FS, which can be used for the same plus for solving the multi-tenancy issue.

The cloud infrastructure is based on clients paying for the exact time of use of a particular service. In a conventional scenario, a container with all of its service and applications is launched, and the client pays for the time of use of the entire container whether he uses certain services or not.

The cloud infrastructure based on targeted virtualization allows the client to pay only for separate container services used by the client. If a service is not virtualized and is needed only from time to time, the client pays less. According to the exemplary embodiment, virtualization switches can be implemented for:

- Files (i.e., system libraries, applications, virtualized file systems/proc and /sys, virtualized blocking utilities, etc.);
- Users and groups (i.e., container own users and groups including root directories);
- Process trees (i.e., when virtualization is on, the container sees only its own processes, beginning with init; process identifiers are also virtualized, thus PID of the init application equals to 1);
- Network (i.e., a virtual network device venet allows the containers to have their own IP addresses, sets of routing rules and firewalls);
- Devices (a server administrator can grant the container access to physical devices, such as network adapters, gateways, disk partitions, etc.); and
- Inter-process communication (IPC) objects (i.e., shared memory, synchronization primitives, messages).

8

In other words, container isolation is fragmented—some of the above components can be virtualized and isolated and some can be shared by other containers.

According to the exemplary embodiment, the targeted (selective) virtualization allows to save a disk space. The OS files do not need to be stored in each of the containers. Only some of the files that are used by the container are stored. A larger number of the containers can be launched on each node (i.e., higher density). The proposed approach to virtualization also provides for more flexible container management by a user or an administrator. The administrator can save resources and efficiently allocate loads created by container applications. Additionally, the administrator can configure only the resources that are going to be virtualized and used, instead of all of the resources.

The clients of the containers that have some non-isolated (i.e., non-virtualized) resources can “see” each other’s resources. This can be advantageous, for example, a common network resource does not require assigning an IP address to each of the containers. It also does not require Network Address Translation. The efficiency of a cloud infrastructure consisting of fragmented containers can be more efficient. For example, disk I/O operations that are not virtualized work faster than the virtualized I/O operations. According to the exemplary embodiment, the cloud infrastructure is the cloud containing the applications (services). The cloud allows to abstract from the OS.

In order to implement an Application Container (i.e., a container with selected virtualization), the configuration file vps.basic can be edited by a text editor as a comment, so it becomes invisible for configuration utility. For example, the section reflecting memory is made invisible. Then, when the container is created, memory quota is not applied, and the container uses the shared memory of the hardware node. According to the exemplary embodiment, the container is launched by the command vzctl start.

This command sets up all network interfaces, initializes VPS quota, if needed, and starts the init process inside the Virtual Private Server (i.e., the container). Note that in case of Linux OS, starting of the init file can be skipped. In other words, Linux utilities are not used in the container. Instead, the configuration procedure only indicates the applications that need to be launched inside the container (i.e., for example, Apache or PHP). In other words, instead of launching the OS resources only selected applications are launched. Thus, the costs of the container operation are reduced and the container-based cloud system becomes more flexible. Note that other OSs can be used.

An example of the command line “vzctl start 100 -apache-virt=on -php-virt=on -perl-virt=off” means that the container with the id=100 has virtualization on for the Apache and PHP applications (i.e., launched in isolation), but Perl is shared with the host (not virtualized). The line “vzctl start 100 -memory-manager=on -pid-virt=off” means that the container with the id=100 has memory virtualization on and the shared pid tree (virtualization is off).

According to the exemplary embodiment, the host OS is not virtualized in a form of a Guest application. The main resources that can be virtualized (or not) are shown in FIGS. 2 and 3. In addition to the described resources the User Beancounters can have a virtualization option. The Beancounters are sets of counters, limits and guarantees for each of the containers. A set of parameters (approximately 20) is used to cover all aspects of container functionality. The parameters are selected in such a way that neither of the containers can use up a resource limited for the entire server (node). Thus,

US 9,348,622 B2

9

the containers cannot hinder the operation of each other. User Beancounters are a set of limits and guarantees controlled per container.

The Parameters are:

Primary Parameters:

numproc—Maximum number of processes and kernel-level threads allowed for this container.

numtcpsock—Maximum number of TCP sockets.

numothersock—Maximum number of non-TCP sockets (local sockets, UDP and other types of sockets).

vmguarpages—Memory allocation guarantee.

Secondary Parameters:

kmemsize—Size of unswappable memory in bytes, allocated by the operating system kernel.

tcpsndbuf—The total size of buffers used to send data over TCP network connections. These socket buffers reside in “low memory”.

tcprecvbuf—The total size of buffers used to temporarily store the data coming from TCP network connections. These socket buffers also reside in “low memory”.

othersockbuf—The total size of buffers used by local (UNIX-domain) connections between processes inside the system (such as connections to a local database server) and send buffers of UDP and other datagram protocols.

dgramrcvbuf—The total size of buffers used to temporarily store the incoming packets of UDP and other datagram protocols.

oomguarpages—The guaranteed amount of memory for the case the memory is “over-booked” (out-of-memory kill guarantee).

privvmpages—Memory allocation limit in pages (which are typically 4096 bytes in size).

Auxiliary Parameters:

lockedpages—Process pages are not allowed to be swapped out
shmpages—The total size of shared memory
physpages—Total number of RAM pages used by processes in a container.

numfile—Number of open files.

numflock—Number of file locks.

numpty—Number of pseudo-terminals.

numsiginfo—Number of siginfo structures.

dcachesize—The total size of dentry and inode structures locked in memory.

numiptent—The number of NETFILTER (IP packet filtering) entries.

swappages—The amount of swap space to show in container, and guarantees controlled per container.

The term Beancounter (user_beancounter or UBC) is a synonym of the CGroups. Each container has its own identifier and virtualization means—the isolation means (name space) and the resource limiting means (i.e., user_beancounters). When the container is created, the Beancounter is defined by two parameters—a threshold and a limit. If the threshold is reached, it is increased by 1. However, the limit cannot be exceeded. In other words, the threshold is needed for informing a container administrator and the limit is used for the actual limitation by the host. Each resource can be seen from the interface “/proc/user_beancounters” or “/proc/bc/<BCID>/resources” as in Application Containers and has five values associated with it: a current usage, a maximum usage (for the lifetime of a container), a threshold, a limit, and a fail counter value. If any resource hits its limit, the fail counter for is the resource is increased. This allows the owner to detect problems in the container by monitoring /proc/user_beancounters in the container.

10

The Beancounter can be viewed as following:

1) An owner or administrator can view the Beancounters on his terminal by using a command “cat/proc/user_beancounters” or “cat/proc/bc/<BCID>/resources” depending on implementation. Note that BCID is Beancounter ID. A table produced by these commands is illustrated in FIG. 6;

2) A special script can be created for checking the state of the Beancounters according to a schedule. The script can analyze the Beancounters’ content and perform the required actions;

3) A container virtualization system monitors the Beancounters in order to limit resource usage.

The resource usage is limited. For example, CPU time is limited. All CPU call from the container processes are converted into a CPU time. If the aggregated CPU time used by the container over a period of time (e.g., in 24 hours) reaches a threshold or exceeds a limit, the scheme of limiting the CPU access by the container processes is implemented. This means that the CPU will process container requests less frequently even the CPU is not busy with other processes. The limits are implemented by the container virtualization technology integrated into the patched Linux kernel. A similar approach can be used with other operating systems as well.

The resource usage can be controlled by a script. For example, the host administrator has created a Beancounter for controlling a number of files in the container. Note that the owner cannot create a Beancounter. A threshold is set at 80,000 files and a limit is 90,000 files. The script reads the data from the Beancounter according to a certain schedule. If the script sees that the number of files is less than 80,000, no actions are taken. If the number of files is larger than 80,000, but smaller than 90,000, the script sends an email notification to the owner of the container indicating that the number of files approaches the limit. If the number of files is larger than 90,000, the script sends a notification the host administrator, who takes appropriate actions for limiting the container files number. The owner can invoke the script at any time using a web interface. The script reads the data out of the beancounter and displays it on the page of the hosting panel.

Thus, the Beancounters are used for logical limitation and real control of resource usage. The user Beancounters can manage a group of processes. The resource usage of these processes is controlled by a set of counters. These counters are controlled by one Beancounter. At the host, one Beancounter monitors the memory and IO together. The Beancounter also monitors aggregate usage of memory and IO by all host processes for billing or for implementing quotas if the limit is exceeded. When the container is launched, a new Beancounter is created for monitoring the usage of memory and IO by the container processes.

In the Application Containers, the container Beancounter can monitor only one parameter. For example, the memory container Beancounter can control memory usage by the container. However the IO can be controlled by the host Beancounter. In a conventional container, this would be impossible—if one makes the Beancounter for the memory only and makes the IO off, then the IO in this container will be consumed uncontrollably.

According to the exemplary embodiment, an OS kernel (Linux or another OS) needs to be patched. The patched (i.e., modified) OS kernel provides for virtualization of selected resources implemented as a separate modules (utilities). If a certain level of virtualization is required by the user of the container, only certain utilities are taken from the OS kernel for creation of the container as opposed to the conventional scenario, where all of the utilities are included into the container. Note that the exemplary embodiment does not prevent

US 9,348,622 B2

11

creation of the conventional containers with a complete set of utilities. The containers can have a shared resource(s). Namely, the network stack can be shared by all of the containers. In this case, the containers use the networking infrastructure implemented on the hardware node. In this implementation, the network virtualization is switched off for all of the containers.

Note that network virtualization cannot be divided into smaller components. Thus, the network stack is always virtualized in a conventional container. However, in the Application Containers as described here, the network stack can remain un-virtualized. Instead, the host network stack is used. In a conventional container, if the network stack is not virtualized, the container does not have a networking capability. In the exemplary embodiment, the container, advantageously, uses the host networking capability.

The proc filesystem entry showing resource control information is the /proc/user_beancounters file inside a container. /proc/user_beancounters on a Hardware Node contains UBC parameters for all containers running on a Node. An example of content of "/proc/user_beancounters" for beancounter with BCID=221 is on FIG. 6.

The Output Contains the Following Fields:

uid—the numeric identifier of a container;
held—current usage of a resource (an accounting-only parameter);

maxheld—an accounting parameter which shows the maximal value of a resource usage during the last accounting period. This period usually matches the container lifetime;

failcnt—the number of refused resource allocations for the whole lifetime of the process group;

barrier—a control parameter which generally defines the normal resource allocation border. For some resources, the parameter can be undefined, while for others, it may be an effectively limiting parameter;

limit—a control parameter which generally defines the top border for a resource allocation. For some resources, the parameter can be undefined, while for others, it may be an effectively limiting parameter.

FIG. 7 illustrates the use of Beancounters in a standard container virtualization schema. A host administrator 703 creates a Beancounter 701, which controls usage of all resources on the host 101. When the container 102 is created, a Beancounter 702 is created for the container with the same set of the resources. Limits for each of the resources are set. In the example depicted in FIG. 7 only two resources, memory and IO are monitored by respective controllers 709 and 710. A container owner 704 can access the resource usage statistics by requesting the Beancounter 702. The container owner 704 can be given the rights (depending on billing solutions) for configuring limits of resource usage via the Beancounters. A rate plan of the owner can change based on the configuration of the limits. The use of memory 712 by all container processes is controlled by a controller 709. A controller 711 controls I/O usage by all the container processes.

FIG. 8 illustrates how the containers are controlled by Beancounters, in accordance with the exemplary embodiment. A host administrator 703 creates the Beancounters 701.1 and 701.2, each of the Beancounters controls one of the host resources. In the example depicted in FIG. 8, the memory is controlled by the controller 705 and the IO is controlled by the controller 706. Note that the Beancounters can be created for all controllable computer resources. According to the exemplary embodiment, when the container 102 is created, it is not necessary to create an entire set of all possible Beancounters. Some of the Beancounters can be omitted. In the example depicted in FIG. 8, the IO control Beancounter is not

12

created. The Beancounter of the container 702 is used for controlling a memory usage by different processes 712. The processes of the container 702 that use IO 711 are monitored by the Beancounter 701.2 on the host. A process, which uses the memory and the IO (e.g., a group of the processes 713) is controlled by the memory controller 709 in the container 102, and is controlled by the IO controller 701.2 on the host 101.

FIG. 9 illustrates how the Beancounters control the processes in accordance with the exemplary embodiment. Each process has one object. The processes #3 and #4 belong to the host. These processes are controlled by all possible Beancounters 701.1-701.n on the host. Each of the Beancounters 701.1-701.n controls a resource used by a process. The processes #101 and #102 belong to the container 102.1. These processes are controlled by the host only for memory and CPU usage. All other resources are virtualized and controlled by the Beancounters 702.1.1-702.1.n of the container 102.1. The processes #201 and #202 belong to the container 102.2. These processes are controlled by the host for network usage only. All other resources are virtualized and controlled by the Beancounters 702.2.1-702.2.n of the container 102.2.

If the virtualization of a particular resource is turned off, the Beancounter is not created in the container. In this case, control of the resource usage by the container is implemented by a host Beancounter as shown in FIGS. 8 and 9. The container processes using a particular resource increase the host counter (not the container counter). According to the exemplary embodiment, the container isolation is less restricting, which makes the container more flexible in terms of use. If virtualization of a container resource is turned off, the resource is not isolated from the same resource of another container.

According to one exemplary embodiment, a selective virtualization/isolation of container resources can be used for optimized data backup. Some containers on the host node can be dedicated to backup operations. The data backup is a procedure that consumes a lot of resources. This procedure is optimized by performing back of only selected container resources. A special container backup utility is used for the backup optimization. The data backup does not require virtualization of a file system and a network stack, because the dedicated container is not intended for a migration. The backup can be performed for a shared file system, for a database or for the entire container. The backup utility is launched on the host system, thus, it can see all host system data. The backup utility turns off the isolation of selected container resources for optimized backup within the container. For example, an isolation of the network stack can be turned off because the backup utility does not need it, since it can use the host network stack.

The isolation of the file system can also be turned off so the backup utility can see the entire host file system and backup only the requested files. However, a limit for memory use isolation cannot be lifted because other host applications need to execute and use the memory. The disk I/O isolation cannot be turned off in the dedicated container, because the backup can take up an entire disk I/O resource. The same goes for isolation of the CPU time. Thus, the backup process is optimized and the host OS does not use additional resources for supporting unnecessary isolation.

With reference to FIG. 5, an exemplary system for implementing the invention includes a general purpose computing device (i.e., a host node) in the form of a personal computer (or a node) 101 or server or the like, including a processing unit 21, a system memory 22, and a system bus 23 that couples various system components including the system memory to the processing unit 21. The system bus 23 may be any of

US 9,348,622 B2

13

several types of bus structures including a memory bus or memory controller, a peripheral bus, and a local bus using any of a variety of bus architectures. The system memory includes read-only memory (ROM) 24 and random access memory (RAM) 25.

A basic input/output system 26 (BIOS), containing the basic routines that help to transfer information between elements within the computer 101, such as during start-up, is stored in ROM 24. The personal computer/node 101 may further include a hard disk drive for reading from and writing to a hard disk, not shown, a magnetic disk drive 28 for reading from or writing to a removable magnetic disk 29, and an optical disk drive 30 for reading from or writing to a removable optical disk 31 such as a CD-ROM, DVD-ROM or other optical media.

The hard disk drive, magnetic disk drive 28, and optical disk drive 30 are connected to the system bus 23 by a hard disk drive interface 32, a magnetic disk drive interface 33, and an optical drive interface 34, respectively. The drives and their associated computer-readable media provide non-volatile storage of computer readable instructions, data structures, program modules and other data for the personal computer 101.

Although the exemplary environment described herein employs a hard disk, a removable magnetic disk 29 and a removable optical disk 31, it should be appreciated by those skilled in the art that other types of computer readable media that can store data that is accessible by a computer, such as magnetic cassettes, flash memory cards, digital video disks, Bernoulli cartridges, random access memories (RAMs), read-only memories (ROMs) and the like may also be used in the exemplary operating environment.

A number of program modules may be stored on the hard disk, magnetic disk 29, optical disk 31, ROM 24 or RAM 25, including an operating system 35 (e.g., WINDOWS™). The computer 101 includes a file system 36 associated with or included within the operating system 35, such as the WINDOWS NT™ File System (NTFS), one or more application programs 37, other program modules 38 and program data 39. A user may enter commands and information into the personal computer 101 through input devices such as a keyboard 40 and pointing device 42.

Other input devices (not shown) may include a microphone, joystick, game pad, satellite dish, scanner or the like. These and other input devices are often connected to the processing unit 21 through a serial port interface 46 that is coupled to the system bus, but may be connected by other interfaces, such as a parallel port, game port or universal serial bus (USB). A monitor 47 or other type of display device is also connected to the system bus 23 via an interface, such as a video adapter 48.

In addition to the monitor 47, personal computers typically include other peripheral output devices (not shown), such as speakers and printers. A data storage device, such as a hard disk drive, a magnetic tape, or other type of storage device is also connected to the system bus 23 via an interface, such as a host adapter via a connection interface, such as Integrated Drive Electronics (IDE), Advanced Technology Attachment (ATA), Ultra ATA, Small Computer System Interface (SCSI), SATA, Serial SCSI and the like.

The computer 101 may operate in a networked environment using logical connections to one or more remote computers 49. The remote computer (or computers) 49 may be another personal computer, a server, a router, a network PC, a peer device or other common network node, and typically includes many or all of the elements described above relative to the computer 101.

14

The computer 101 may further include a memory storage device 50. The logical connections include a local area network (LAN) 51 and a wide area network (WAN) 52. Such networking environments are commonplace in offices, enterprise-wide computer networks, Intranets and the Internet. When used in a LAN networking environment, the personal computer 101 is connected to the local area network 51 through a network interface or adapter 53.

When used in a WAN networking environment, the personal computer 101 typically includes a modem 54 or other means for establishing communications over the wide area network 52, such as the Internet. The modem 54, which may be internal or external, is connected to the system bus 23 via the serial port interface 46. In a networked environment, program modules depicted relative to the personal computer 101, or portions thereof, may be stored in the remote memory storage device. It will be appreciated that the network connections shown are exemplary and other means of establishing a communications link between the computers may be used.

Having thus described the different embodiments of a system and method, it should be apparent to those skilled in the art that certain advantages of the described method and apparatus have been achieved. In particular, it should be appreciated by those skilled in the art that the proposed method provides for efficient use of container resource and for more flexible cloud infrastructure.

It should also be appreciated that various modifications, adaptations, and alternative embodiments thereof may be made within the scope and spirit of the present invention. The invention is further defined by the following claims.

What is claimed is:

1. A system for targeted virtualization in containers, the system comprising:

- a host hardware node having a host OS kernel;
- a plurality of host OS kernel objects implemented on the host hardware node; and
- at least one container running on the host hardware node, the container virtualizing the host OS and using selected host OS kernel utilities,

wherein:

- the host OS kernel utilities have a virtualization on-off switch;
- the selected host OS kernel object is virtualized inside the container if the utility virtualization switch is on; and
- the container uses the host OS kernel objects shared with other containers running on the hardware node.

2. The system of claim 1, wherein the utility virtualization switch is turned on by a container administrator.

3. The system of claim 1, wherein the utility virtualization switch is turned on based on a container user requirements.

4. The system of claim 1, wherein the host OS kernel objects are not virtualized.

5. The system of claim 1, wherein the host OS kernel objects are shared among containers running on the hardware node.

6. The system of claim 1, wherein the container uses selected virtualized host OS kernel objects and shared host OS kernel objects based on user requirements.

7. The system of claim 1, wherein the host OS kernel is patched for selected virtualization of utilities in a form of separate utility modules.

8. The system of claim 1, wherein the host OS kernel objects used for selected virtualization are any of:

- memory;
- I/O operations;
- disk;

15

network;
users and groups;
devices;
PID tree;
IPC objects;
user applications; and
system modules.

9. The system of claim 1, wherein the container is a part of a cloud-based infrastructure.

10. A computer-implemented method for targeted virtualization in a container, the method comprising:

- (a) launching an OS kernel on a host hardware node;
- (b) patching the host OS kernel for selected virtualization of host OS kernel objects;
- (c) activating a utility virtualization on-off switch on the host OS kernel objects;
- (d) selecting the host OS kernel objects to be virtualized by turning the utility virtualization switch on;
- (e) launching a first container on the hardware node, the first container virtualizing the host OS;
- (f) virtualizing the selected host OS kernel objects inside the container;
- (g) repeating steps (b)-(f) for another container launched on the hardware node, wherein the first container shares the host OS kernel objects that are not virtualized with other containers running on the hardware node.

11. The method of claim 10, wherein the selecting of the host OS kernel objects is implemented based on user requirements.

12. The method of claim 10, wherein the containers cannot access objects virtualized inside the other containers.

13. The method of claim 10, wherein the containers form a cloud infrastructure.

16

14. The method of claim 10, further comprising creating a beancounter for controlling resource usage by container processes.

15. The method of claim 14, wherein virtualized container resources are controlled by the beancounter.

16. A computer-implemented method for backup optimization in a dedicated container, the method comprising:

- (a) launching an OS kernel on a host hardware node;
- (b) patching the host OS kernel for selected virtualization of host OS kernel objects;
- (c) activating a utility virtualization on-off switch on the host OS kernel objects;
- (d) selecting the host OS kernel objects to be virtualized by turning the utility virtualization switch on;
- (e) launching a dedicated backup container on the hardware node, the backup container virtualizing the host OS;
- (f) virtualizing the selected host OS kernel objects inside the dedicated container; and
- (g) starting a host backup utility configured to backup container data,

wherein the host backup utility only backs up the host OS kernel objects that are not virtualized within the dedicated backup container.

17. A system for targeted virtualization in a container, the system comprising:

- a processor;
- a memory coupled to the processor; and
- a computer program logic stored in the memory and executed on the processor, the computer program logic for implementing the steps (a)-(g) of claim 10.

* * * * *

EXHIBIT C

US007461148B1

(12) **United States Patent**
Belousov et al.(10) **Patent No.:** **US 7,461,148 B1**
(45) **Date of Patent:** **Dec. 2, 2008**(54) **VIRTUAL PRIVATE SERVER WITH
ISOLATION OF SYSTEM COMPONENTS**(75) Inventors: **Serguei M Belousov**, Herndon, VA
(US); **Stanislav S Protassov**, Singapore
(RU); **Alexander G Tormasov**, Moscow
(RU)(73) Assignee: **SWsoft Holdings, Ltd.** (BM)(*) Notice: Subject to any disclaimer, the term of this
patent is extended or adjusted under 35
U.S.C. 154(b) by 1187 days.(21) Appl. No.: **10/703,594**(22) Filed: **Nov. 10, 2003****Related U.S. Application Data**(63) Continuation-in-part of application No. 09/918,031,
filed on Jul. 30, 2001, now Pat. No. 7,099,948.(60) Provisional application No. 60/467,547, filed on May
5, 2003, provisional application No. 60/269,655, filed
on Feb. 16, 2001.(51) **Int. Cl.**

G06F 15/16	(2006.01)
G06F 15/173	(2006.01)
G06F 15/167	(2006.01)
G06F 9/45	(2006.01)
G06F 9/46	(2006.01)
G06F 13/28	(2006.01)
G06F 7/38	(2006.01)
G06F 3/00	(2006.01)
G06F 3/048	(2006.01)
G06F 9/44	(2006.01)
G06F 13/00	(2006.01)
G06F 9/00	(2006.01)
G06F 17/00	(2006.01)
G06F 12/00	(2006.01)
G06F 9/455	(2006.01)
G06F 15/00	(2006.01)

(52) **U.S. Cl.** **709/226**; 709/215; 709/229;
719/319; 711/152; 712/227; 703/22(58) **Field of Classification Search** 709/203,
709/215, 219, 226, 229, 236; 711/152, 153,
711/173; 718/1; 719/319; 726/15; 715/778;
712/227; 703/22

See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,761,477 A * 6/1998 Wahbe et al. 718/1
(Continued)

FOREIGN PATENT DOCUMENTS

JP 2004274448 A * 9/2004

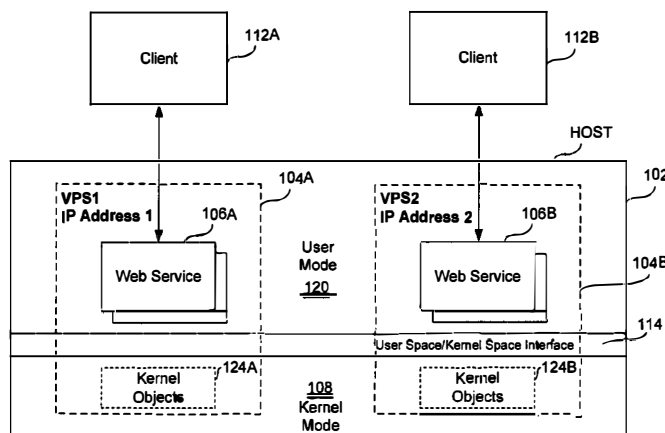
OTHER PUBLICATIONS

Killalea, T. "Recommended Internet Service Provider Security Ser-
vices and Procedures," RFC 3013, Nov. 2000, pp. 1-13.*

(Continued)

Primary Examiner—Jason D Cardone*Assistant Examiner*—Melvin H Pollack(74) *Attorney, Agent, or Firm*—Bardmesser Law Group(57) **ABSTRACT**

A server includes a host running an operating system kernel. Isolated virtual private servers (VPSs) are supported within the kernel. At least one application is available to users of the VPS. A plurality of interfaces give the users access to the application. Each VPS has its own set of addresses. Each object of each VPS has a unique identifier in a context of the operating system kernel. Each VPS is isolated from objects and processes of another VPS. Each VPS includes isolation of address space of each user from address space of a user on any other VPS, isolation of server resources for each VPS, and failure isolation. The server includes a capability of allocating (or reallocating) system resources to a designated VPS, allocating (or reallocating) system resources to a VPS in current need of such resources, dynamically allocating (or reallocating) VPS resources to a VPS when additional resources are available, and compensating a particular VPS in a later period for a period of under-use or over-use of server resources by the particular VPS in a current period. VPS resources are allocated for each time cycle. All the VPSs are supported within the same OS kernel.

80 Claims, 9 Drawing Sheets

US 7,461,148 B1

Page 2

U.S. PATENT DOCUMENTS

5,905,990 A	5/1999	Inglett	
6,075,938 A *	6/2000	Bugnion et al.	703/27
6,269,409 B1 *	7/2001	Solomon	719/329
6,332,180 B1 *	12/2001	Kauffman et al.	711/153
6,374,286 B1 *	4/2002	Gee et al.	718/108
6,381,682 B2 *	4/2002	Noel et al.	711/153
6,385,643 B1 *	5/2002	Jacobs et al.	709/203
6,397,242 B1 *	5/2002	Devine et al.	718/1
6,460,082 B1 *	10/2002	Lumelsky et al.	709/226
6,496,847 B1 *	12/2002	Bugnion et al.	718/1
6,542,926 B2 *	4/2003	Zalewski et al.	709/213
6,560,613 B1	5/2003	Gylfason et al.	
6,601,110 B2 *	7/2003	Marsland	719/310
6,618,736 B1	9/2003	Menage	
6,633,916 B2 *	10/2003	Kauffman	709/229
6,647,508 B2 *	11/2003	Zalewski et al.	714/3
6,681,310 B1 *	1/2004	Kusters et al.	711/202
6,687,762 B1 *	2/2004	Van Gaasbeck et al.	719/319
6,697,876 B1 *	2/2004	van der Veen et al.	719/313
6,732,211 B1 *	5/2004	Goyal et al.	710/261
6,754,716 B1 *	6/2004	Sharma et al.	709/238
6,772,419 B1 *	8/2004	Sekiguchi et al.	719/319
6,802,063 B1 *	10/2004	Lee	718/1
6,816,941 B1 *	11/2004	Carlson et al.	711/111
6,845,387 B1 *	1/2005	Prestas et al.	709/203
6,854,009 B1 *	2/2005	Hughes	709/220
6,862,650 B1 *	3/2005	Matthews et al.	711/6
6,886,165 B1 *	4/2005	Muller et al.	719/310
6,907,421 B1 *	6/2005	Keshav et al.	707/2
6,912,221 B1 *	6/2005	Zadikian et al.	370/395.21
6,941,545 B1 *	9/2005	Reese et al.	717/130
6,957,237 B1 *	10/2005	Traversat et al.	707/206
6,976,054 B1 *	12/2005	Lavian et al.	709/203
6,976,258 B1 *	12/2005	Goyal et al.	718/104
6,985,937 B1 *	1/2006	Keshav et al.	709/223
6,996,828 B1 *	2/2006	Kimura et al.	719/319
7,035,963 B2 *	4/2006	Neiger et al.	711/6
7,099,948 B2 *	8/2006	Tormasov et al.	709/229
7,143,024 B1 *	11/2006	Goyal et al.	703/21
7,219,354 B1 *	5/2007	Huang et al.	719/328
7,225,441 B2 *	5/2007	Kozuch et al.	718/1
7,266,595 B1 *	9/2007	Black et al.	709/223
7,275,246 B1 *	9/2007	Yates et al.	718/100
7,356,817 B1 *	4/2008	Cota-Robles et al.	718/1
2002/0065917 A1 *	5/2002	Pratt et al.	709/226
2002/0112090 A1 *	8/2002	Bennett et al.	709/319

OTHER PUBLICATIONS

Barham, Paul et al. "Xen and the Art of Virtualization," Proceedings of the ACM Symposium on Operating Systems Principles (SOSP), Oct. 19-22, 2003, pp. 1-14.*

Banga, Gaurav et al. "Resource Containers: A New Facility for Resource Management in Server Systems," USENIX Association Third Symposium on Operating Systems Design and Implementation (OSDI), 1999, pp. 45-58.*

Zhang, Wensong. "Linux Virtual Server for Scalable Network Services," Ottawa Linux Symposium, 2000, pp. 1-10.*

* cited by examiner

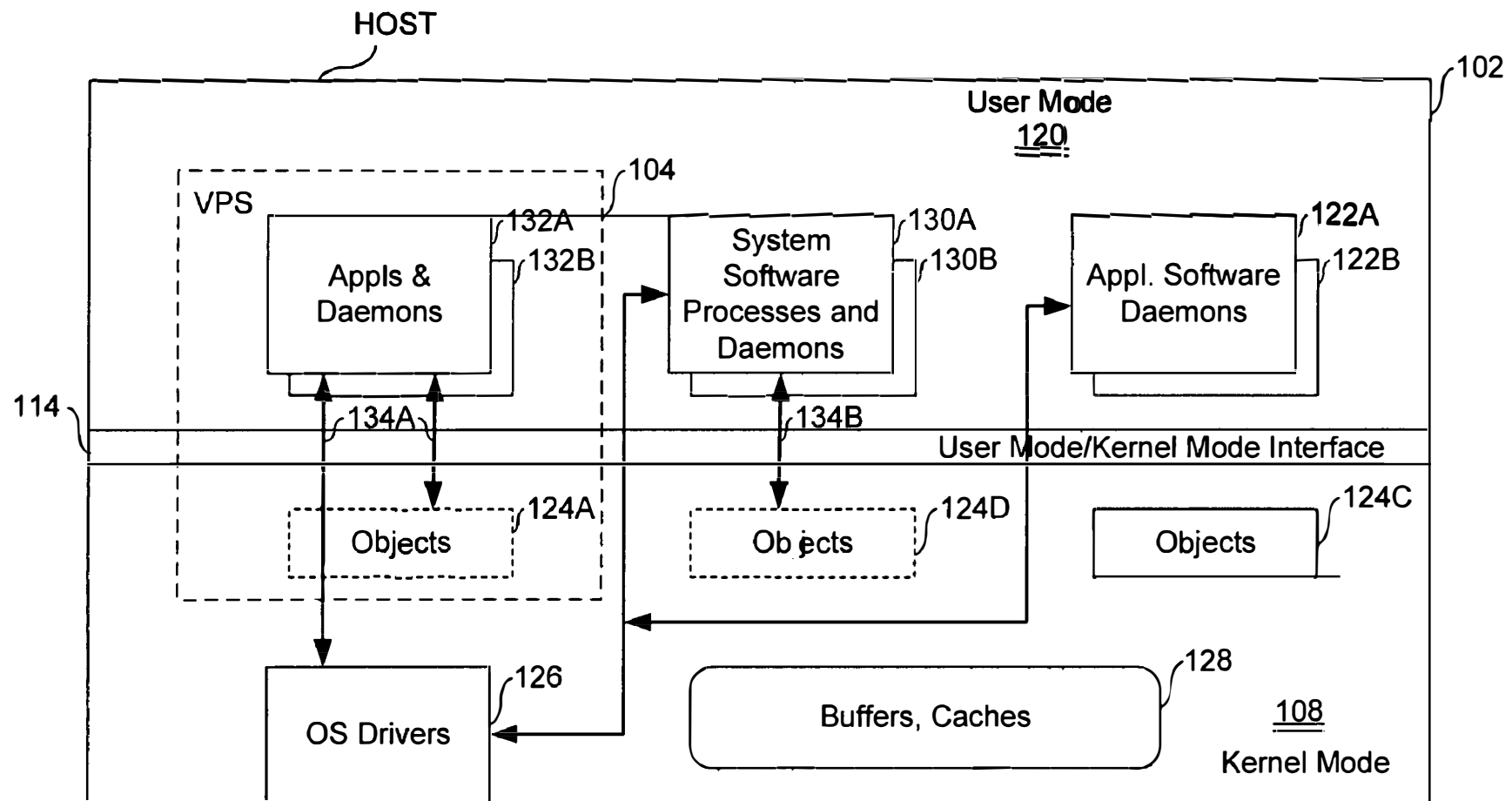


FIG. 1A

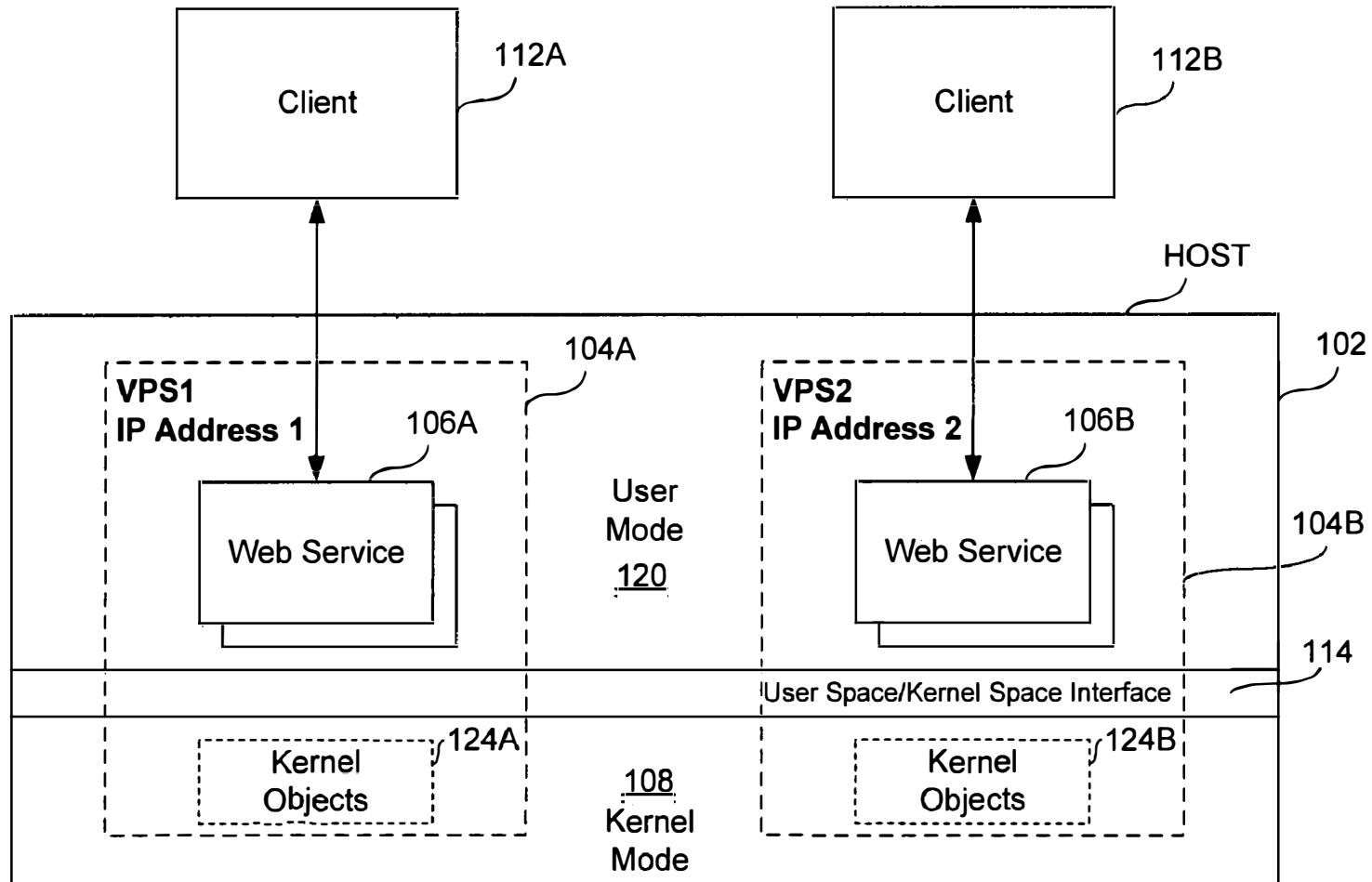


FIG. 1B

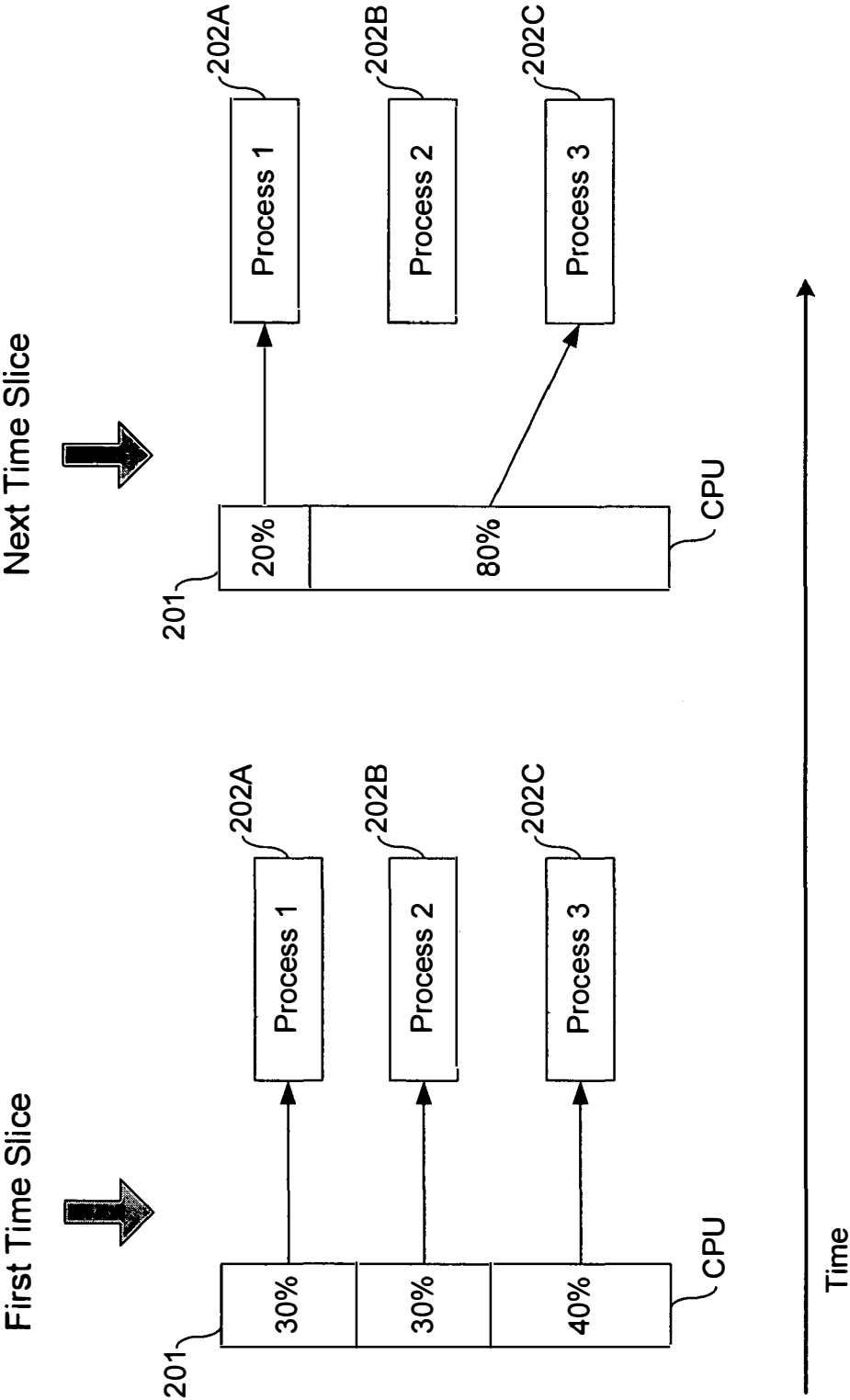


FIG. 2

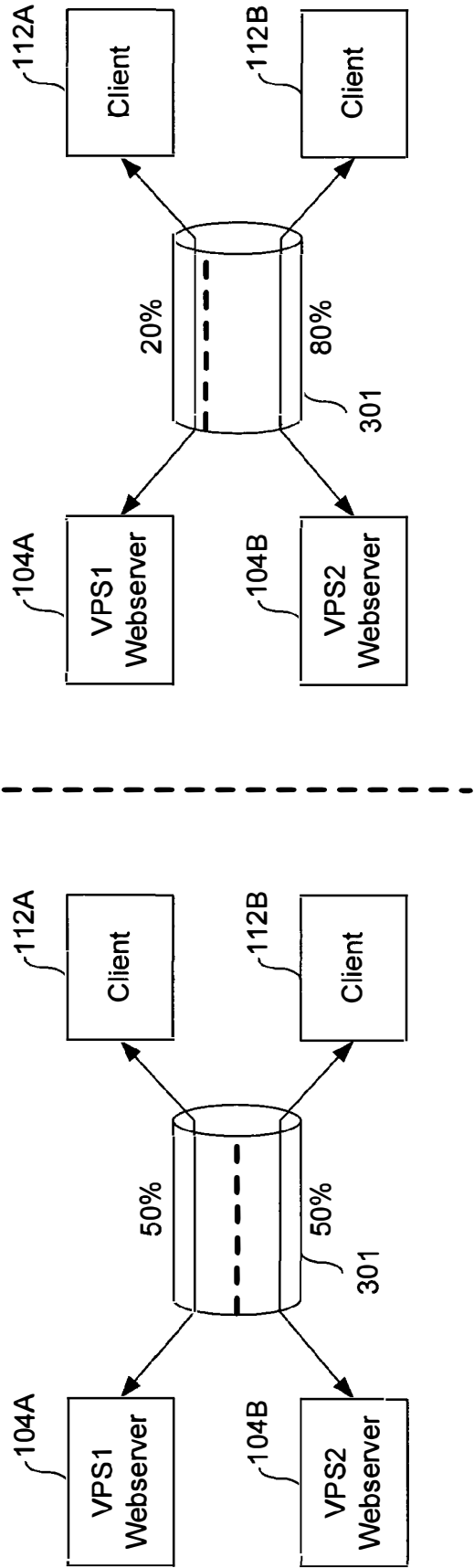


FIG. 3

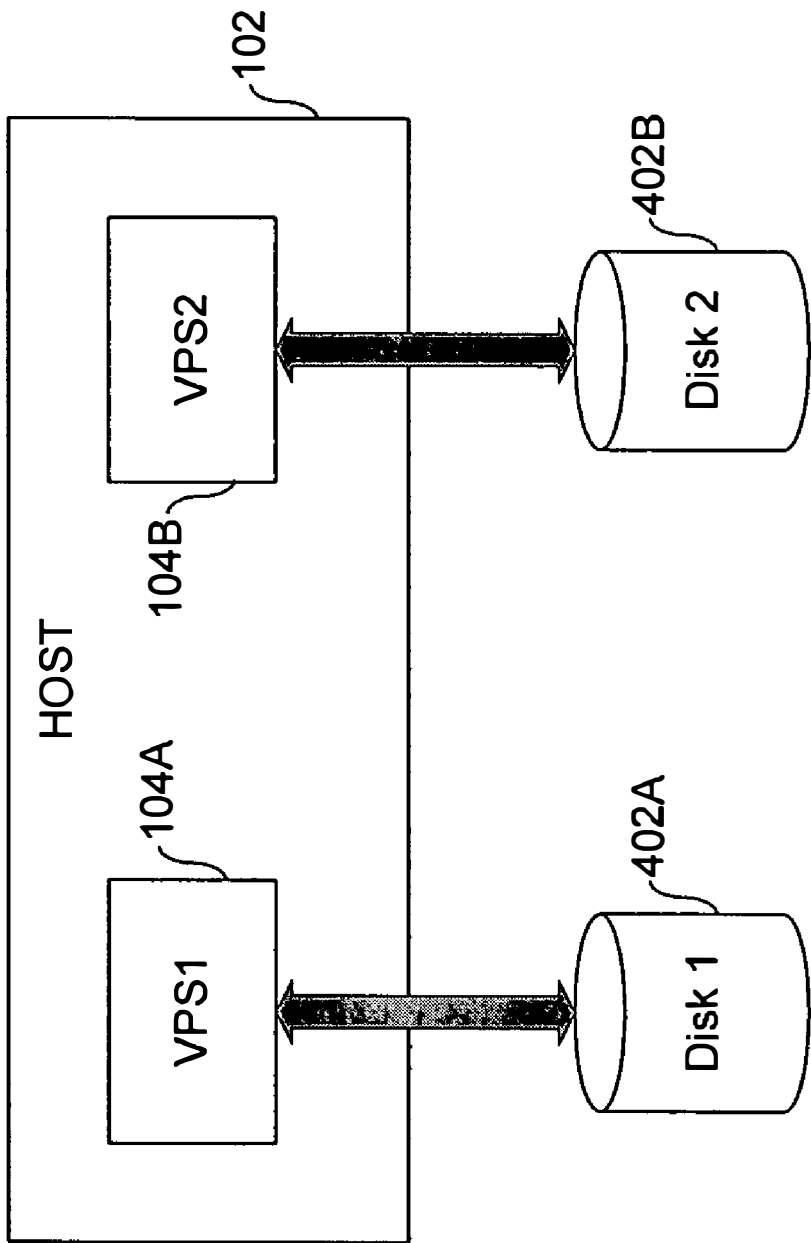


FIG. 4

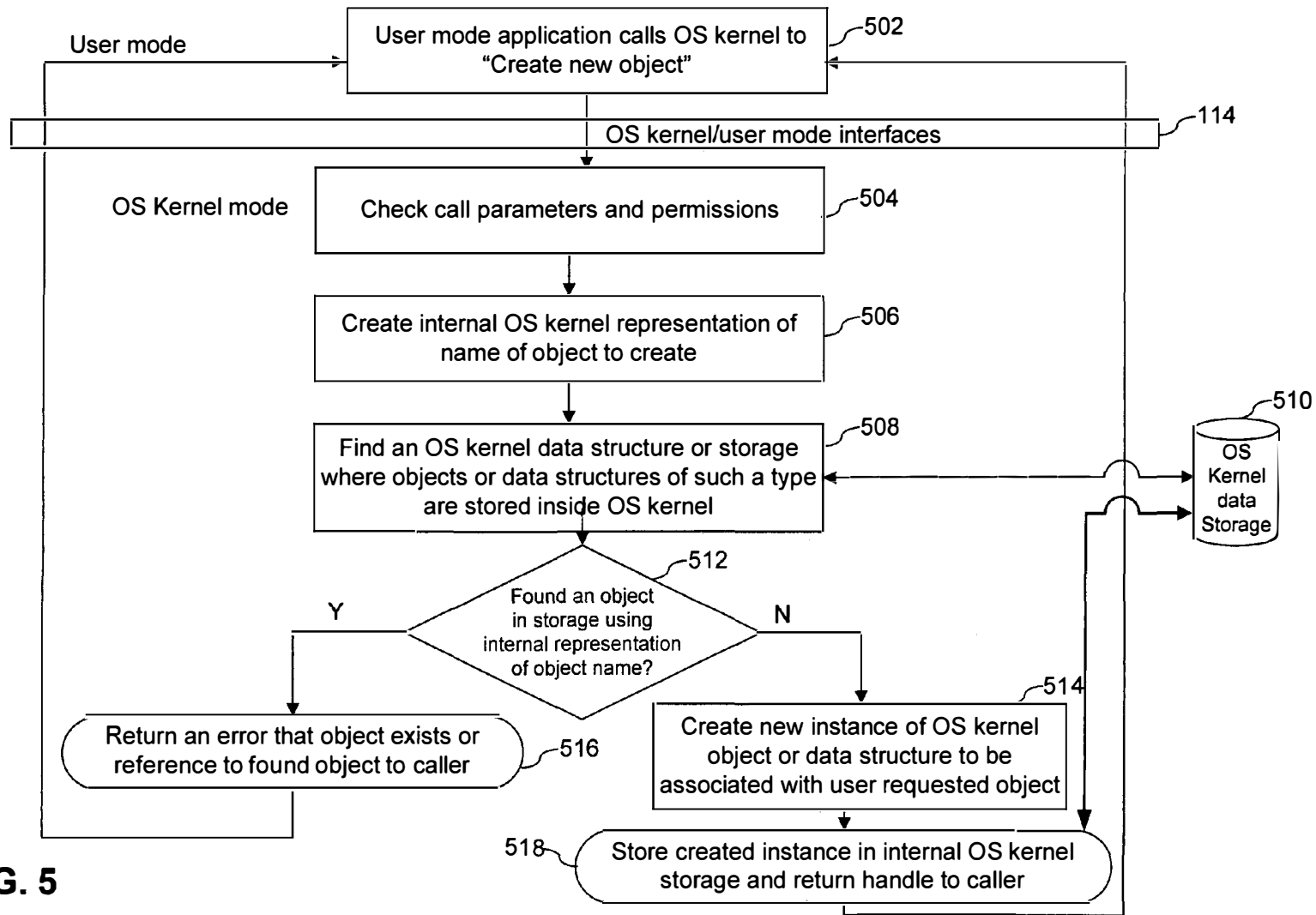
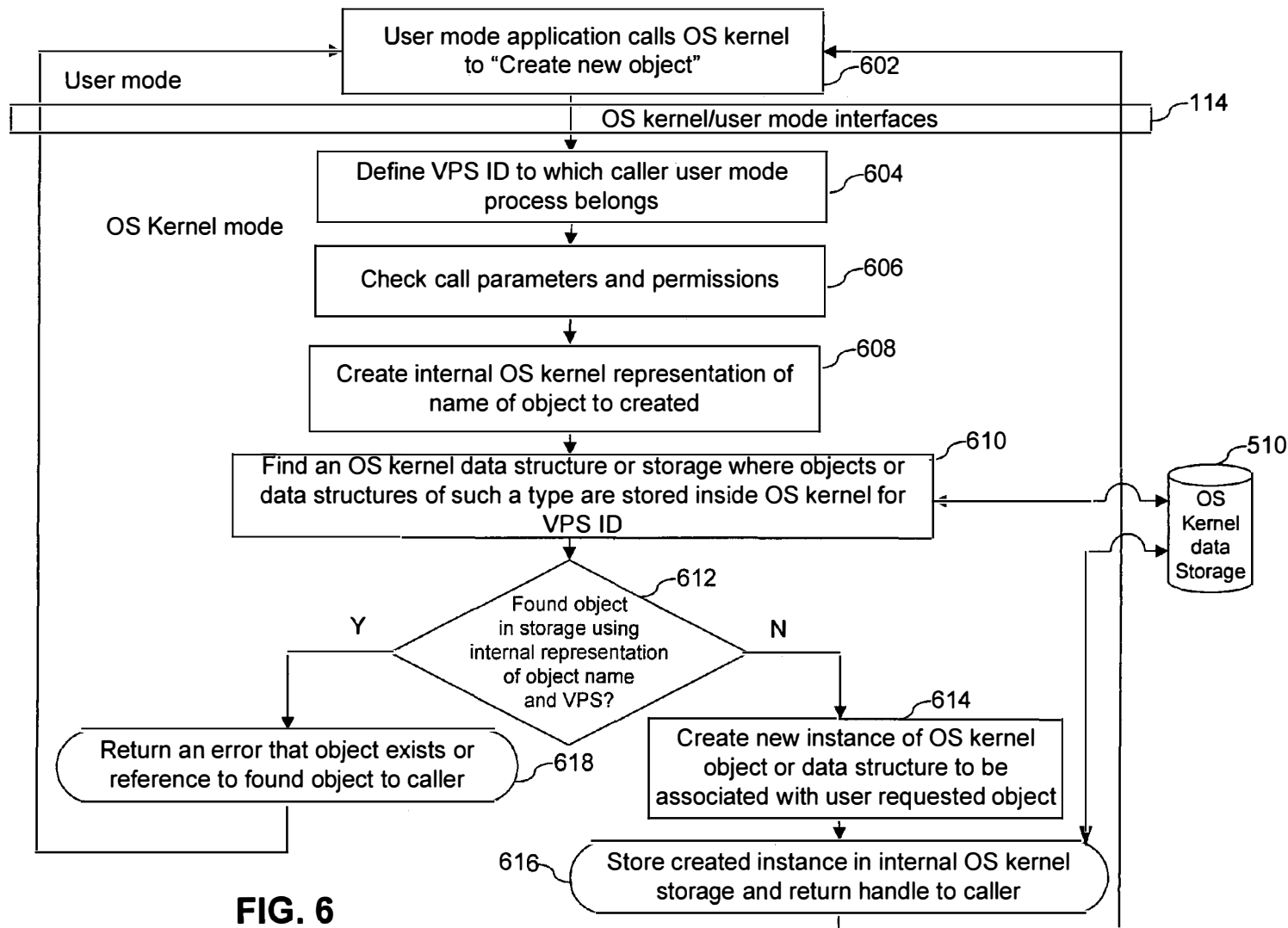


FIG. 5



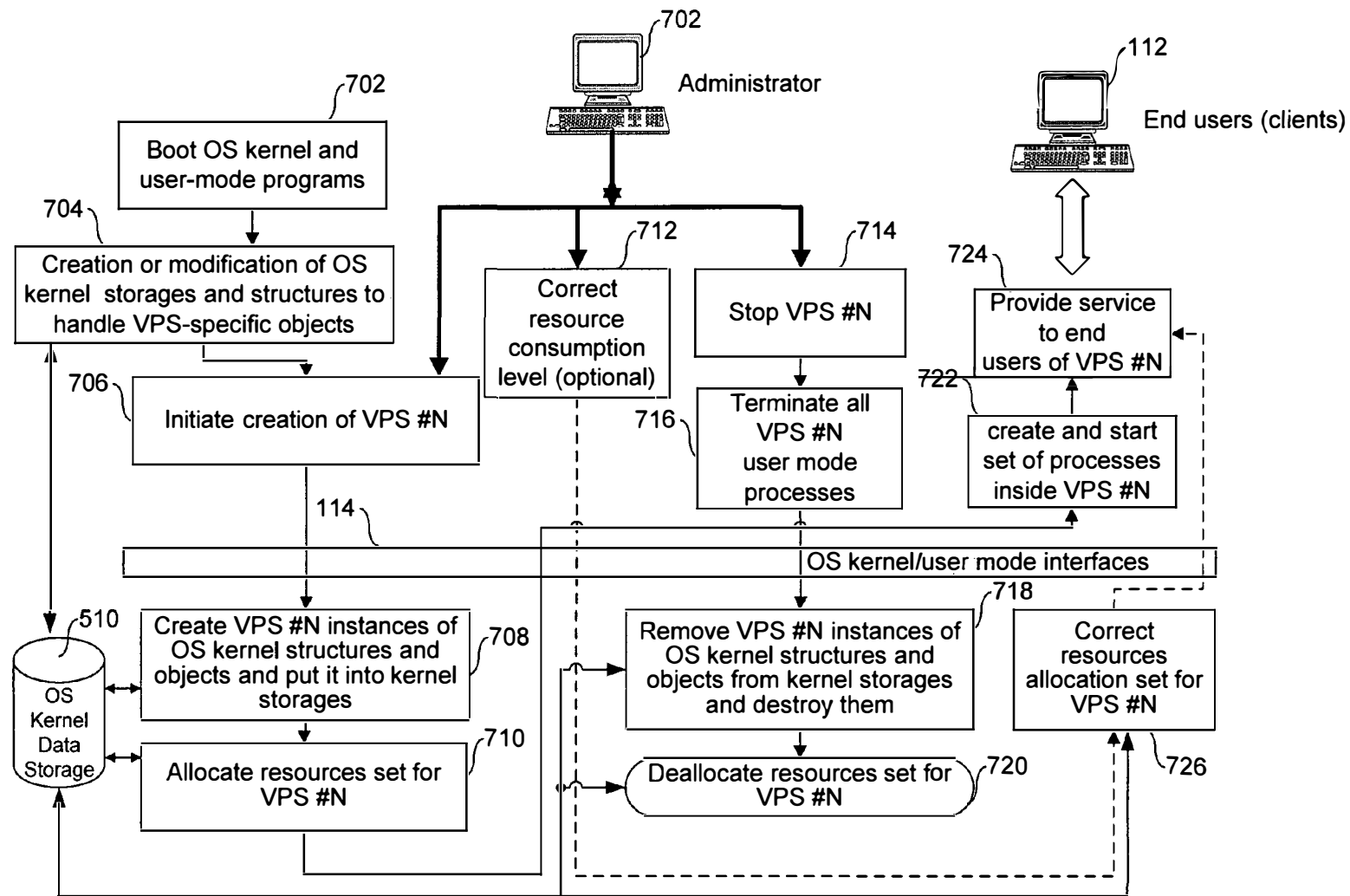


FIG. 7

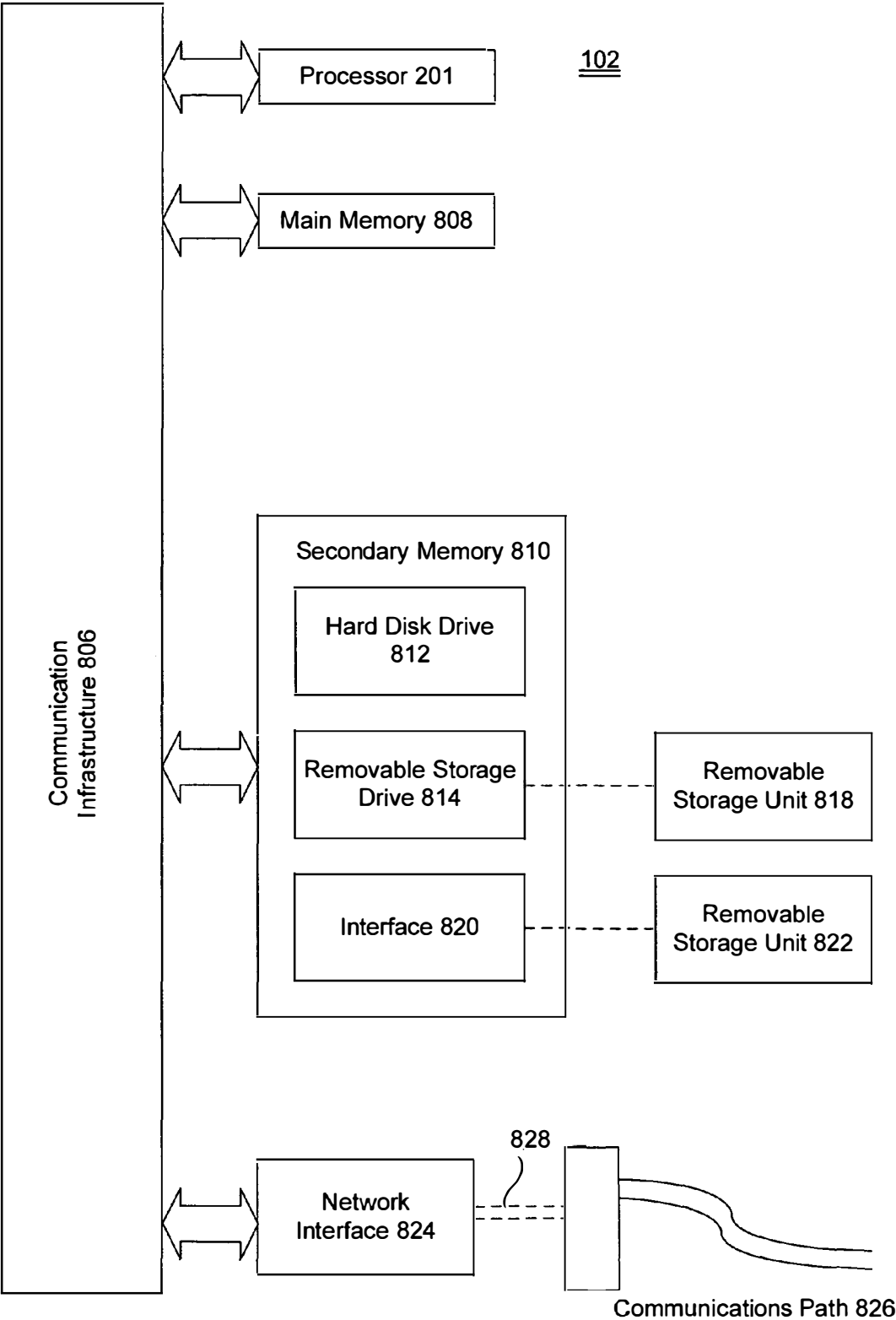


FIG. 8

US 7,461,148 B1

1

VIRTUAL PRIVATE SERVER WITH ISOLATION OF SYSTEM COMPONENTS

CROSS-REFERENCE TO RELATED APPLICATIONS

This application is a continuation-in-part of U.S. patent application Ser. No. 09/918,031, filed on Jul. 30, 2001, which is a non-provisional of U.S. Provisional Patent Application No. 60/269,655, filed on Feb. 16, 2001, and to U.S. Provisional Patent Application No. 60/467,547, filed on May 5, 2003, which are all incorporated by reference herein.

BACKGROUND OF THE INVENTION

1. Field of the Invention

The present invention is related to virtual private servers, and more particularly, to isolated virtual private servers that appear to a user as a stand-alone server.

2. Related Art

With the popularity and success of the Internet, server technologies are of great commercial importance today. An individual server application typically executes on a single physical host computer, servicing client requests. However, providing a unique physical host for each server application is expensive and inefficient.

For example, commercial hosting services are often provided by an Internet Service Provider (ISP), which generally provides a separate physical host computer for each customer on which to execute a server application. However, a customer purchasing hosting services will often neither require nor be amenable to paying for use of an entire host computer. In general, an individual customer will only require a fraction of the processing power, storage, and other resources of a host computer.

Accordingly, hosting multiple server applications on a single physical computer is desirable. In order to be commercially viable, however, every server application needs to be isolated from every other server application running on the same physical host. Clearly, it would be unacceptable to customers of an ISP to purchase hosting services, only to have another server application program (perhaps belonging to a competitor) access the customer's data and client requests. Thus, each server application program needs to be isolated, receiving requests only from its own clients, transmitting data only to its own clients, and being prevented from accessing data associated with other server applications.

Furthermore, it is desirable to allocate varying specific levels of system resources to different server applications, depending upon the needs of, and amounts paid by, the various customers of the ISP. In effect, each server application needs to be a "virtual private server" or VPS, simulating a server application executing on a dedicated physical host computer.

Such functionality is unavailable on traditional server technology because, rather than comprising a single, discrete process, a virtual private server must include a plurality of seemingly unrelated processes. Each process performs various elements of the sum total of the functionality required by the customer. Because each virtual private server includes a plurality of processes, traditional server technology has been unable to effectively isolate the processes associated with one virtual private server from those processes associated with other virtual private servers.

Another difficulty in implementing multiple virtual private servers within a single physical host involves providing each server with a separate file system. A file system is an orga-

2

nized accumulation of data within one or more physical storage devices, such as a hard disk drive or RAID (redundant array of inexpensive disks). The data is typically organized into "files," such as word processing documents, spreadsheets, executable programs, and the like. The files are stored within a plurality of "storage units" of the storage device, sometimes referred to as "disk blocks" or "allocation units."

Unfortunately, providing a separate physical device for storing the file system of each virtual private server would be expensive and inefficient. Accordingly, it would be desirable to store the file systems of multiple virtual private servers within the same physical device or comparatively small set of devices.

Thus, a major problem with conventional VPS implementations is the lack of isolation between the VPSs. This means that a conventional VPS has to operate in a "friendly environment," relying on other VPSs and other applications running in those other VPSs to not invade its address space, or to utilize more than their share of resources. This is also sometimes known as a cooperative environment (vs. non-cooperative environment, where users or applications of one VPS cannot be trusted to not modify data that does not belong to them or to not attempt to "hog" all system resources). However, there is a difficulty of utilizing "cooperative" VPSs in any number of applications. For example, in the web server context, it is assumed that the host will be subject to attack by hackers. No assumption of a friendly environment can be made in that case. Also, in the absence of isolation between the VPSs, one VPS can "hog" more than its share of system resources, or can affect and/or modify objects and data that belong to other VPSs.

Accordingly, there is a need for an effective way of isolating the VPSs from one another in a server environment.

SUMMARY OF THE INVENTION

Accordingly, the present invention is related to a virtual private server with isolation of server components that substantially obviates one or more of the disadvantages of the related art.

In one embodiment, there is provided a server including a host running an operating system kernel. A plurality of isolated virtual private servers (VPSs) are supported within the operating system kernel. At least one application is available to users of the VPS. A plurality of interfaces give the users access to the plurality of applications.

Each VPS has its own virtual address space (or its own set of addresses), which includes memory, IP addresses, disk drive addresses, SMB network names, TCP names, pipe names, etc. Each VPS has its own objects and data structures. Each of the objects and the data structures of each VPS has a unique identifier in a context of the operating system kernel. Each VPS cannot affect data structures of another VPS, or objects of another VPS, and cannot access information about processes running on another VPS. Each VPS includes isolation of address space of each user from address space of a user on any other VPS, isolation of server resources for each VPS, and isolation of application program failure effects on any other VPS. The server resources include any of a virtual memory allocated to each user, a pageable memory allocated in the OS kernel to support the VPSs, a pageable memory used by the OS kernel for support of user processes either in shared, or in exclusive form (i.e., either in support of user processes of one VPS, or in support of user processes of multiple VPSs), a resident memory allocated in the OS kernel, physical memory used by the user processes, a share of CPU resources, security descriptors (or other identifiers

US 7,461,148 B1

3

related to the rights of the users and the VPSs), objects and data structures used by the OS kernel, I/O interfaces and their utilization level by the particular VPS, file and/or disk space, and individual user resource limitations.

Each VPS typically includes a plurality processes, each with at least one thread servicing corresponding users, a plurality of objects associated with the plurality of threads, a set of user and group IDs that unique in the context of a VPS corresponding to users and groups of a particular VPS, a set of configuration settings corresponding to each VPS stored within the VPS and a corresponding set of configuration settings for all VPSs stored by the kernel, a unique file space, means for management of the particular VPS, means for management of services offered by the particular VPS to its users, and means for delivery of the services to the users of the particular VPS.

The server includes a capability of allocating any of the system resources to a designated VPS, a capability of allocating any of the system resources to a VPS in current need of such resources, a capability of dynamically allocating any of the VPS resources to a VPS when additional resources are available, and a capability of compensating a particular VPS in a later period for a period of under-use or over-use of server resources by the particular VPS in a current period. The server can force a VPS return allocated resources to a common pool of resources, or the VPS may relinquish the resources allocated to it. The server defines a set of time cycles, such that VPS resources are allocated or reallocated for each such time cycle. The server dynamically partitions and dedicates resources to the VPSs based on a pre-established service level agreement. All the VPSs are supported within the same OS kernel. Some functionality of the VPSs can be supported in user space.

The server software can be an add-on to any of Microsoft Windows NT Server—Terminal Server Edition, Microsoft Windows 2000 Server—Terminal Server, and Microsoft Windows Server 2003—Terminal Services, or any server based on a Microsoft Windows product. The operating system includes a plurality of threads for execution of user requests. The VPSs appear to a user as substantially stand-alone servers, and generally provide the functionality of a stand-alone server or remote computer, including all administrative operations.

Additional features and advantages of the invention will be set forth in the description that follows, and in part will be apparent from the description, or may be learned by practice of the invention. The advantages of the invention will be realized and attained by the structure particularly pointed out in the written description and claims hereof as well as the appended drawings.

It is to be understood that both the foregoing general description and the following detailed description are exemplary and explanatory and are intended to provide further explanation of the invention as claimed.

BRIEF DESCRIPTION OF THE ATTACHED DRAWINGS

FIGS. 1A-1B show a system block diagram of one embodiment of the present invention.

FIG. 2 illustrates an example of CPU resource management.

FIG. 3 illustrates an example of dynamic partitioning of resources in a context of bandwidth allocation.

FIG. 4 illustrates an example of resource dedication.

FIG. 5 illustrates one method of object creation in a VPS.

FIG. 6 illustrates another method of object creation.

4

FIG. 7 illustrates a life cycle of a VPS according to the present invention.

FIG. 8 illustrates an example of a host architecture that may be used in the present invention.

DETAILED DESCRIPTION OF THE INVENTION

Reference will now be made in detail to the preferred embodiments of the present invention, examples of which are illustrated in the accompanying drawings.

The present invention is directed to a system, method and computer program product for creating and managing virtual private servers or VPSs. A VPS is a closed set, or collection, of processes, system resources, users, groups of users, objects and data structures. Each VPS has an ID, or some other identifier, that distinguishes it from other VPSs. The VPS of the present invention offers to its users a service that is functionally substantially equivalent to a standalone server with remote access. From the perspective of an administrator, the VPS of the present invention appears substantially the same as a dedicated computer at a data center. For example, the administrator of the VPS of the invention has the same remote access to the server through the Internet, the same ability to reload the server, load system and application software (except modifications of the OS kernel and loading arbitrary kernel drivers or modules, and direct access to the hosts' physical hardware without support from the VPS implementation), launch VPS users, establish disk space quotas of the users and user groups, support storage area networks (SANs), set up and configure network connections and web servers, etc. In other words, substantially the full range of system administrator functions are available as if the VPS were a dedicated remote server, with the existence of the VPS being transparent from the perspective of both the VPS user and the VPS system administrator. From the user perspective, the VPS functionally acts essentially like a remote server, and offers the same services, for example, through a dedicated IP address. Note that some processes running in kernel mode can provide services to multiple VPSs. Note also that due to locality of particular resources and data related to a particular VPS, the present invention lends itself to supporting migration of a VPS to another physical host (e.g., another host in a cluster) by transfer of all data related to the VPS from one physical host to another. See U.S. Provisional Patent Application No. 60/467,547, filed on May 5, 2003, which is incorporated by reference herein.

This approach has a number of advantages compared to conventional approaches, particularly in the fields of web-hosting and server enterprise consolidation. The system administrator does not require any special training, since the VPS functions essentially identically to a standalone server, and the administrator operations are the same. Furthermore, the launch of multiple VPSs in the same host permits a higher level of server utilization. With effective VPS resource isolation, it is possible to offer a guaranteed level of resource availability specified in a service level agreement (SLA).

It should be noted that, although there is no standard terminology, in a multi-process environment such as being described herein, processes (sometimes called "actors") generally are thought of as including one or more threads and related data (as well as non-thread related data), and threads can have sub-threads or fibers. It will be appreciated that this discussion is intended to be descriptive only, and the invention is not limited to the specific terminology, since and different operating systems use different terms for threads (sometimes, for example, also called "lightweight processes").

US 7,461,148 B1

5

One embodiment of the present invention includes a physical computer (usually called a “host”) that is configured to run multiple isolated virtual private servers (VPSs). The host includes an operating system (OS) kernel. The kernel runs a number of processes and threads for execution of various system related processes. The host also includes a number of processes running on the server that correspond to the VPSs, with typically at least one user connected to at least one VPS.

The host can include: a physical memory, such as a dynamic random access memory (RAM) used by the various processes and threads. A virtual memory is allocated to each user or kernel process. A memory is allocated to the kernel of the operating system (either physical, pageable or non-pageable memory), and to various objects and data structures used by the operating system. The system may also include various storage elements, such as caches, and operating system buffers. The computer system also includes a central processing unit (CPU), or optionally multiple CPUs, whose time may be shared between and among the VPSs. A number of peripheral devices may be included, such as disk storage devices, input/output interfaces, network interfaces, etc. The storage mechanisms, particularly disk storage, may include a number of data files, databases, metadata storage and various other permanently and temporarily stored data objects.

Generally, as used in this description, the operating system kernel has one or more “processes” running within inside kernel mode. The operating system itself may be viewed as a process. The OS process has its own objects for its own internal use, as well as a number of objects that are representative of the VPSs supported within it.

Each VPS is therefore a set of processes running within user mode and having support from the operating system kernel. Each such VPS typically has a number of OS users and groups, with objects representative of the users associated with the particular VPS. Also, each VPS typically has a number of processes and threads that correspond to, for example, application software run by a particular OS user. Each of these processes can also have multiple threads (sub-threads) running within them, sometimes called “fibers.”

Each of the OS users typically has a unique identifier. For example, each user may have an identifier that is unique in the context of its own VPS, and which may be, but not necessarily, unique in the context of the host. Alternatively, each VPS may own user IDs, such that each user has a globally unique identifier within the host or domain of computers.

The VPS also includes a number of interfaces that permit the users to access the various services available on the server and on the kernel. Such interfaces include system calls, shared memory interfaces, I/O driver control (ioctl), and similar mechanisms. The operating system kernel includes a number of execution threads, lightweight processes, and/or other primitives for execution of the services to the users and for servicing user requests. Each VPS typically has its own “virtual address space” that may be partitioned among its users.

The OS kernel also typically uses a number of objects and data structures, each of which has an identifier that is unique within the context of the operating system kernel. Such identifiers are sometimes known as handles or descriptors (or sometimes they can simply be kernel memory addresses), depending on the particular OS implementation.

The operating system kernel also includes a number of processes, threads, lightweight processes, or other primitives running inside kernel address space in kernel mode. Typically, the kernel has its own virtual address space, and includes a number of objects and structures servicing the OS kernel processes. The kernel also may include a number of

6

structures and objects that service the processes run by the users. The kernel may also include mechanisms for management and enumeration of operating system and VPS objects and structures, when such objects and structures are utilized by the OS kernel itself and by various processes run by the users and/or by the VPSs.

Each VPS typically includes a number of processes, threads, lightweight processes, and other primitives for servicing the users of that particular VPS. Each VPS also typically has its own objects and data structures that are associated with the processes and threads of that particular VPS. Each VPS may also include a number of objects and data structures utilized by the operating system for control of that VPS. Each VPS also may include its own set of OS users and groups, each of which has a unique ID in the context of that particular VPS, but, as noted above, not necessarily unique in the context of the entire host or other VPSs. Each VPS also preferably includes its own file/disk space, which is allocated by the kernel for exclusive use by that VPS.

The VPS typically offers a number of services to the users. Examples of such services may be database access, webpage access, use of application software, remote procedure calls, etc.

Each VPS offers a number of services to users of that VPS, which are implemented as processes within that VPS. From the perspective of the user, the existence of the VPS is transparent, such that to the user it appears that he has an entire remote server dedicated to himself.

The system of the present invention will be further described with the aid of FIGS. 1A and 1B. Here, FIG. 1A is intended to illustrate the general case, and FIG. 1B, the application of the present invention to a webserver context. A host 102 is running an operating system with a kernel 108. Two VPSs 104A and 104B are shown, each with their own IP address space. Within each VPS 104, a web server is running (here, designated by 106A, 106B, respectively). It will be appreciated that the web server example is used only for illustration purposes, and any remote server functionality may be implemented. Also, in FIGS. 1A and 1B, 114 designates an operating system interface (for example, typically including system calls, I/O controls (ioctl/fcntl), drivers, etc., which are used for communication between user space (user mode) 120 processes and kernel space (kernel mode) 108 processes). In this context, the OS kernel 108 is the protected part of the operating system, typically the part that is essential to the functioning of the host 102. Also, other, non-critical, applications may be loaded (e.g., the “Minesweeper” game in Microsoft Windows is a well known example of such a non-critical application that is typically a part of the loaded operating system, but is not a critical part).

122A, 122B are a set of applications and daemons running in user mode 120. The server also includes a number of application programs, system services, daemons, and similar constructs typically utilized by the users (for example, Microsoft Word, database software, webpage servers, remote procedure calls, support daemons, Telnet servers, etc.). 124A is a set of objects and data structures associated with a process that runs in the user mode 120, even though the objects and data structures 124A themselves exist in the kernel mode 108.

124A, 124B designate a set of objects associated with a particular process. 124D designates a set of objects and data structures associated with operating system software processes. 124C designates a set of objects and structures that are used by the OS kernel 108 itself, for its own purposes. 126 designates operating system hardware drivers. 128 designates OS kernel buffers, caches, and other similar structures that are used for storage, and typically that have enumeration ability.

US 7,461,148 B1

7

130A, 130B are system software processes and daemons. 132A, 132B are applications and daemons that exist in user space 120. Other examples of daemons include, for example, web server daemons that interact with Internet Explorer, RPC port mapper servers (i.e., servers that do not directly interface with users), etc.

134A, 134B designate communications between user space 120 processes and kernel space 108 processes. In the present invention, the isolation between the VPSs 104 is done at the kernel level. Some of the VPS 104 functionality may be implemented in user space 120.

The web servers 106A, 106B are connected to users (clients), in this case one client each, designated as 112A and 112B, respectively. Here, the client 112 is an example of a user. Another example of a user may be a VPS administrator. As noted above, the kernel 108 has a set of objects (here labeled as kernel objects 124A, 124B) that are representative of the VPS 104A, 104B, respectively. The client 112A can connect to the web server 106A using an IP address 1. The client 112B can connect to the web server 106B through an IP address 2. These are isolated, i.e., non-shared, addresses. The objects and structures associated with VPS 104A are distinct and isolated from the objects and structures associated with VPS 104B.

Additionally, each VPS 104A, 104B has its own set of resources, for example, its own disk space, file space, its own share of common network adapter bandwidths, etc. The failure or error of web server 106A will not influence or interfere with the activities of VPS 104B. Thus, the client 112B can continue accessing its web server 106B even in the event of program failure or crash on the VPS 104A side. It will be appreciated that although the example of FIG. 1B is in only terms of two VPSs 104A, 104B, in actuality, any number of VPSs 104 may be running on the host 102 (e.g., thousands), limited only by physical system parameters, CPU speed, bandwidth and other resources limitations.

The amount of system resource use may be regulated by the operating system kernel 108, particularly by establishing limits on system resource use for each VPS 104. In particular, the operating system kernel 108 allows reserving a particular system resource for a particular VPS 104. Various algorithms may be used to insure that only actual users of a particular system resource are allocated that resource, thus avoiding allocating a share of system resources, to those VPSs 104 that do not have users utilizing those resources at that particular time. Furthermore, the operating system kernel 108 dynamically allocates resources for the VPSs 104. Optionally, the resources allocated to a particular VPS 104 may exceed its originally (for example, SLA-established) limits, when such resources are available to be utilized, assuming other VPSs 104 are not utilizing those resources. The resource allocation mechanism also allows compensation for over-use or under-use by a particular VPS 104 in a particular time period (time slice). Thus, if one VPS 104 under-utilizes its allocated system resources during one time slice, it may be allocated a greater share of system resources during the next time slice, and vice versa.

As noted above, each VPS 104 has a share of system resources allocated to it. The resources allocated to each VPS 104 (for example, memory, file space, I/O, etc.) are isolated from each other through the use of a VPS ID and other specific kind of addressing. In the absence of active user processes utilizing the relevant resources (in other words, when other VPSs 104 are under-utilizing their resources), another VPS 104 can take advantage of that by utilizing more than its originally allocated share of system resources if permitted by host configuration settings.

8

FIG. 2 illustrates an example of resource management, as discussed above. As shown in FIG. 2, a number of user processes, labeled 202A, 202B, 202C are running on a CPU 201, with the percentage of CPU resources allocated as shown –30%, 30%, and 40%, respectively, at time slice 1. As time progresses and the next time slice begins, process 202B is inactive, and process 1 requires only 20% of CPU utilization during that particular slice. Accordingly, process 202C is able to utilize the remaining 80% of the CPU resources. This may be referred to as a soft limit if the user running process 202C is allotted 40% of CPU time under his service level agreement. If the 40% figure is a “soft upper limit,” the operating system kernel 108 will allocate the entire remaining 80% of the CPU resources to process 202C. On the other hand, if the 40% limit is a “hard limit,” then the CPU usage by process 202C would remain at 40%. A similar approach may be taken with almost any system resource, for example, network bandwidth, disk space, memory usage, etc.

FIG. 3 illustrates an example of dynamic partitioning of resources in a context of bandwidth allocation. In FIG. 3, 301 designates a communication channel, for example, an Internet link. Initially, both web servers 106A, 106B are connected to their respective user processes (clients) 112A, 112B, sharing the communications link 301, with 50% of the bandwidth allocated to each. If the service level agreement for user process 112B includes a soft limit on bandwidth then, when user process 112A only requires 20% of the bandwidth of the channel 301, the entire remaining 80% of the bandwidth can be dynamically allocated to user process 112B.

FIG. 4 illustrates an example of resource dedication. As shown in FIG. 4, the host 102, running two VPSs 104A, 104B may be connected to two disk drives 402A, 402B. Each VPS 104 is allocated a dedicated disk drive. These may be either physical disk drives or virtual disk drives, such that each VPS 104A, 104B can only access its own disk drive 402A, 402B, respectively, and is unaware of the existence of any other VPS's disk drive.

To enable the resources allocation mechanism to each VPS 104, time slices (or time cycles) are defined such that resources are allocated on a time slice basis and (if necessary) reallocated during the next time slice. Different time slices can be defined for different resource classes. The system determines the amount of resources necessary for each VPS 104 to perform its function prior to the beginning of the next time slice. The system resources may be dynamically partitioned and dedicated, so as to ensure that the resources are allocated in accordance with the established level of service level agreement (SLA) for each VPS 104.

FIG. 5 illustrates one approach to the life-cycle of a VPS object using kernel mode interfaces 114. As shown in FIG. 5, a user mode application calls the OS kernel 108 to create a new object (step 502). The OS kernel 108 checks the call parameters and permissions associated with the calling process (step 504). An internal OS kernel representation of the object is created, including a name of the object about to be created (step 506). A kernel data structure is identified where objects or data structures of this type are stored for use by the OS kernel 108 (step 508). The process then checks whether it found an object in storage 510 using the kernel representation of the object name (step 512). The storage 510 may be used both by the kernel 108 and by the VPSs 110. If yes, then an error is returned to the calling process, indicating that such an object already exists (step 516) and the process then returns to step 502 (depending on the parameters of the call in step 502).

If the object is not found, a new instance of the OS kernel object is created, to be associated with the object requested by the calling process (step 514). The instance of the name (or

US 7,461,148 B1

9

similar object ID) object is stored in the storage **510**, and a handle (i.e., an identifier) is returned to the caller (step **518**). The process then returns to step **502** and the requesting (calling) process continues its execution.

Thus, in the conventional approach, the operating system, upon creation of a user process, or upon response to an existing user process, can create internal objects and data structures that are meant for storage of service information associated with the requesting user process. For example, upon creation of an instance of a process, it is usually necessary to create a special table, which later is used for storage of open process handles and their associated system objects. For example, files and sockets, which this user process can utilize for its functioning, may need to be created. These objects may be unique for a particular process, or they may be associated with a number of different processes simultaneously (for example, the image of an executable file stored in a cache may be used by different processes). The objects themselves must be stored in the data structures of the operating system kernel, in order for the operating system to be able to remove them, add to them, search them, etc. This is the purpose of the storage **510** in FIG. **5**.

The storage **510** may be, for example, dynamic random access memory, caches, buffers, files, etc. Other examples of storage **510** include a hash table to store IP addresses that are related to network interfaces, a linked list of processes initialized in the operating system kernel, a cache of pages associated with a disk (disk read and disk write cache), virtual memory descriptors for particular processes that correspond between virtual pages and physical pages in the memory, and swap file descriptors.

Step **506**, as described above, means that the operating system kernel, when servicing a system call, must be able to work with a representation of an object name within the kernel. This representation (which may be referred to as a "name") may be, for example, similar in some sense to the original. For example, it can be a complete file name. It may also be, for example, an internal memory address that corresponds to a data structure that is responsible for a particular user object.

With further reference to step **516**, the exact nature of what is returned to step **502** depends on the call options. Thus, for example, it may be that in response to an attempt to create a file with an existing name, a message will be returned that such a file cannot be created because it already exists, giving the user an option to overwrite it. As an alternative, the user may be prompted with a question of whether he wants to open the existing file.

FIG. **6** illustrates the creation of objects **124A** according to the present invention, which permits isolation between the various VPSs **104**. As shown in FIG. **6**, a service **106** running in the user mode **120** calls the kernel **108** to create a new object (step **602**). In the OS kernel mode **108**, the VPS ID to which this process belongs is identified (step **604**). Call parameters and permissions of the caller are checked based on the VPS ID and optionally other conditions (step **606**). An internal representation within the OS kernel **108** is created, including the name of the object (step **608**). A storage **510** is searched for OS kernel data structure, where objects or data structures of this type are stored for that particular VPS ID. If such an object is found (step **612**) an error is returned that such an object exists (step **618**). If no such object is found a new instance of such an object **124A** is created, such that it will be associated with the user request (step **614**). An instance of the object name is stored in the storage **510**, and a handle (or object id) is returned to the caller (step **616**). The process then proceeds back to step **602** to continue execution.

10

The user **112** (e.g., a client or a VPS administrator) also has various mechanisms for controlling the VPS **104**, for example, system administrator privileges that extend only to the administration of that particular VPS **104**. The VPS **104** also has means for delivering to the user processes the results of the work performed in response to user requests, such as delivery of information, webpages, data from databases, word processing documents, system call results other files, etc. The VPSs **104** of the present invention permits isolation of each VPS **104** from other VPSs **104** running on the same physical host **102**. This is achieved through a number of mechanisms, as described below:

An address isolation of user services allows specifying different addresses for the different services that are located in different copies of the VPSs **104**. Each service or application launched with the VPS **104** should preferably be individually addressable. Users of that VPS **104** should be able to select an address and identify the server located within that particular VPS **104**. A typical example of such an address is an IP address of the webserver (or DNS name), or network SMB name in MS Network, used for access of MS Windows shared resources, or a telnet server address, used for interactive login. When choosing a particular address, the user must be certain that he will be accessing the server associated with that IP address, and no other, and that someone trying an address that does not belong to that VPS **104** will not be accessing that VPS **104**.

Another example of address isolation is isolation of credentials necessary for system login and authentication. In that case, even after entering a login of the VPS **104**, the user will only be authenticated for that VPS **104**, where he exists as a user. Anywhere else in the host **102**, trying to access an incorrect IP address will result in service denial.

Isolation of all objects is implemented at the operating system kernel level, which insures that no process running on one VPS **104** can take advantage of objects running on any other VPS **104**. This includes an inability by any user processes or any VPS **104** to renumber any object that does not belong to that particular VPS **104** or to effect a change of the state of any such object that does not belong to that particular VPS **104**. This is necessary in order to prevent a malicious user from using application software to access another VPS **104**. An example of such an attempt is a possibility of "killing a process" on another VPS **104**. In this case, the user calls a special function in the kernel **108** (syscall), which can force the process kill, and whose function call parameter is the identifier of the process ("pid") to be killed. Without effective object isolation, a malicious user can kill processes on other VPSs **104**. A similar concept applies to other objects and structures of the kernel **108**, which are function call parameters of an API used within an application running inside a VPS **104**. Also, file systems used by the VPS **104** need isolation to avoid being accessed by a malicious user on a different VPS **104**. Thus, the invention prevents the use of system-offered APIs by one VPS's application to affect objects and structures associated with another VPS **104**.

System resource isolation is necessary to ensure fulfillment of SLA conditions and guarantees. Each VPS **104**, upon creation, has a set of resources allocated to it by the kernel **108**. Examples of such resources are disk space, CPU time, memory use, etc., which may be used by processes within each VPS **104**. The primary goal of such isolation is prevention of "hogging" of system resources by one VPS's process or processes.

Such hogging could prevent other VPSs **104** from delivering a satisfactory level of service to their users. This can occur accidentally, due to an error in a particular application, or

US 7,461,148 B1

11

deliberately, when one user attempts a Denial of Service attack. These resources may be external-oriented (e.g., network traffic and bandwidth), or host-specific, such as cache space dedicated by the kernel 108 to this process. A part of resource isolation is therefore ensuring a minimum level of service and minimum speed of execution of the user processes.

Additionally, failure isolation is also provided between processes running on one VPS 104 and processes running on other VPSs 104. This includes isolation from application crashes, webserver crashes and various other failure modes to which such processes are susceptible. Failure isolation implies that the same process cannot serve users of different VPSs 104. For example, with conventional shared webhosting, all users receive an ability to use different virtual web-servers that are nevertheless hosted by the same actual (“real”) webserver host. In this case, if a user of one web-server gains access to the host or gains exclusive access to its resources (e.g., through a CGI script), then all other users of all other virtual web-servers will be unable to receive service. A badly written CGI script can easily use up significant CPU resources (for example, approaching 100% of the CPU resources), leaving almost nothing for other users. An example of a badly written CGI script is an infinite loop, as follows:

```
line 1
goto line 1
```

Failure isolation is in part obtained through resource isolation, although it also includes the functioning of the server as a whole, beyond just resource utilization by a particular process or user.

In the present invention, failure isolation is realized at the kernel level. In the conventional approach, failure isolation is usually done in the user space through application code checks and library changes. However, a malicious user can change the code of his own process in order to defeat the isolation, since he has access to his own code. This permits, for example, a Denial of Service attack.

Another conventional approach uses special user processes launched in user space. This has a severe overhead impact, since each OS call goes through a lengthy process of various additional calls in the different address spaces of different user mode processes.

Another way of effecting isolation is through hardware emulation in the user process, which can launch an operating system shell. An example of such an approach is an emulator from VMWare (see, e.g., http://www.championsg.com/Champions_InnSite_v10.nsf/pages/Vmware). This approach ensures isolation, but results in additional overhead of having a “kernel above a kernel”—one real kernel, one emulated one. RAM use is highly inefficient in this case, since each kernel requires its own physical memory space. In contrast, the present invention provides the advantages of low overhead and high efficiency of RAM utilization, combined with scalable and effective isolation of the VPS processes from other VPSs.

It should be noted that the isolation of the VPSs 104 from each other as described above is accomplished through the use of objects and data structures of the operating system kernel 108. Support of multiple VPSs 104 can be implemented within a single OS kernel 108, further enhancing the isolation of the operating system to isolate the VPSs 104 from each other.

FIG. 7 illustrates the process of a life cycle of a VPS 104 according to the present invention. As shown in FIG. 7, the OS kernel 108 and various user mode programs are booted up (step 702). OS kernel storages and structures that handle

12

VPS-specific objects (see 510 in FIG. 6) are either created and/or modified (step 704). A VPS #N with an ID number N is created (step 706). Instances of OS kernel objects and data structures corresponding to the VPS #N are created and placed in storage (step 708). A set of resources are allocated for the VPS #N (step 710). Processes are started and/or create inside the VPS #N (step 722). The VPS #N provides services to the end user 112 (step 724). The host system administrator has the option of initiating creation of the VPS #N (step 706), correction of resource consumption level (step 712) and of the resource allocation level (step 726) or to stop a VPS #N (step 714). Upon stopping of the VPS #N, user mode processes for that VPS are terminated (step 716). All instances of OS kernel objects and structures corresponding to VPS #N are deleted or removed (step 718). All resources allocated to VPS #N are de-allocated (step 720).

It should be noted that identification of the call context, which identifies which particular VPS 104 made the request to create an object, is an important aspect of object isolation. Without it, the operating system typically will be unable to determine whether a particular action is or is not allowed for the particular call. In the present invention, the operating system creates a data structure that stores the information corresponding to the VPS 104, which usually occurs when the VPS itself is created. Some of the information may be stored permanently. For example, in the file system there may be a configuration file for each VPS. The data structure also typically includes a number of parameters used for the functioning of the VPS 104 itself. Preferably, there is a short way of naming the VPSs 104, such that searching through the data structure can be accomplished rapidly. This may be a VPS ID, which can be, for example, a number, an alphanumeric string, or similar constructs. Those objects that are unambiguously identified as associated with a particular VPS 104 typically cannot be reassigned with another VPS, since such a possibility usually suggests a concurrent possibility of a security hole. In other words, if a process 132 is “born” in the context of one VPS 104, it cannot then live in another VPS.

The process of context identification of a particular call usually needs to define which VPS made the call, in other words, from which context this call originated. If the call to create an object 124A is generated from within user space 120, then to identify the VPS context, it is only necessary to identify the corresponding process ID, and through the process ID, it is possible to determine the VPS ID to which that process 132 belongs. If the call to create an object 124A is generated within the operating system kernel 108 itself and has no apparent user initiating that call (for example, receipt of a packet from a network adapter, which needs to be identified through its IP address), then the method of its identification depends on the context of the call and is typically defined by a special algorithm, which is dependent on a particular implementation.

Determination of a VPS ID that corresponds to the process is also implementation-specific, and may be done through adding to each process structure a VPS ID at the moment a process is created. This is similar to session ID in Microsoft Windows Terminal Server. Alternatively, there may be a list of how process IDs correspond to VPS IDs, where the process IDs are globally unique. In other words, each process has a globally unique ID in the context of the entire host. As yet another option, process IDs may be duplicated. In a sense, that process IDs are unique only within the context of a particular VPS, but not necessarily globally unique within the entire host 102. Then, a different mechanism would be necessary for determining the call context (instead of relying on the process ID).

US 7,461,148 B1

13

In the embodiment shown in FIG. 7, it is generally assumed that all processes that are “born” from a process 132 that has been previously launched within a particular VPS 104, belong to that VPS 104.

Normally, a VPS 104 is launched by launching a special process, with which a particular VPS ID is associated. After that process with the initial ID starts, other processes, which service that particular process, can be launched, such that they will themselves inherit the VPS ID of that particular parent process.

The present invention offers a significant advantage from the perspective of a system administrator of the host. Because of the various isolation capabilities, the effective utilization of the physical resources of the host may be significantly enhanced. This ultimately results in a lower total cost of ownership (TCO) in an enterprise or data center operation. Such a lower cost of ownership may be due to a reduced need to purchase additional hardware or hardware upgrades, less overhead associated with space rental, air conditioning, power, etc. There may also be lower network costs, and lower administrative overhead from installation or training. Administration of upgrades may also be less expensive because of similarity of user environments and other reasons, and customer satisfaction may be greater due to less apparent system failure and better ability by the data center to adhere to the service level agreement guarantees. Additionally, VPS owners can easily delegate some administration duties to data center administrator.

In one embodiment of the present invention, the VPSs 104 may be implemented as a software expansion of capabilities of Windows-based servers (i.e., “add-ons”), for example, Microsoft Windows NT servers, Microsoft Windows 2000 servers, Microsoft Windows server 2003, and various derivatives thereof, by adding the missing isolation mechanisms and VPS resource management mechanisms to the operating system kernels of those operating systems.

In order to implement the VPS 104 of the present invention, the following steps typically need to be followed. First, the operating system needs to be installed initially on the computer. The software supporting the function of the VPSs 104 needs to be installed including, if necessary, various modules and programs residing within the operating system kernel. Additionally, various service modules and daemons, which function in user space 120, also may need to be installed. The operating system needs to be configured for support of the VPSs 104, for example, by installation of optional templates for subdividing the virtual address space and the file space between the various VPSs 104. The operating system may optionally need to be rebooted. Additional interfaces may need to be provided (such as system calls, ioctls, and other such resources) in order to enable client access to the various application software modules and other operating system functions that are normally accessible to the clients, and/or that enable the functionality of the VPSs 104.

The VPS functionality for each VPS 104 typically includes creation of a corresponding file structure and files to be used by the VPS 104. These may include, e.g., administrative files, user files, etc. Information relating to the particular VPS 104 is registered in a registration database and various other similar structures maintained by the operating system. This information is intended to enable continued functioning of the VPS 104. Additionally, each VPS 104 may be allocated its own IP address or group of IP addresses and other network resources.

The VPS 104 is then launched, after which the operating system starts running a number of processes or threads corresponding to the users 112 of that particular VPS 104. User

14

process requests are then serviced by the host 102, with these requests being passed through the VPS 104 to the operating system kernel 108. Upon termination or shutdown of a particular VPS 104, all threads and processes associated with that VPS 104 are also terminated.

In general, in order to implement the present invention, the address space is divided into two areas: kernel space 108, which is used by the operating system in kernel mode, and is normally is not accessible by application software run by the users 112, and a set of virtual address spaces dedicated to user processes, generally referred to as “user space,” 120, in which the VPSs 104 and the user applications 132 running in them exist. Each VPS 104 has its own set of addresses used by user processes during to access different data locally and across a network. This is done in order to prevent security violations by a VPS 104. In the present invention, some of the VPS functionality may exist in the kernel space 108. Some of the functionality can also exist in the user space 120. In the user space 120, there may be software 130 supporting the needs of the operating system kernel. An example of such functionality may be a daemon that gathers statistics about the VPS 104 use. Another example may be a daemon that monitors VPS 104 processes, etc.

As noted above, each VPS 104 has a number of objects associated with it. Such objects are representative of the users 112, corresponding user processes and/or the applications being run within that VPS 104. Examples of such objects are file descriptors, security tokens, graphical objects (for example, used by graphical software to represent images), etc. Examples of objects 124D within the operating system kernel 108 are file descriptors, etc. Here, a data structure may be thought of as a special (simple) case of an object that does not include functional aspects associated with the object.

The VPS ID discussed above may be also thought of as similar to a mark or a handle, or a similar identifier or structure data structure that can be used for VPS 104 tracking. Thus, the isolation is done through the tracking of VPS 104 and the objects associated with that VPS 104 by a VPS ID.

Although the description above is in terms of a single host 102 running multiple VPSs 104, the invention is equally applicable to a server cluster, where multiple hosts 102 are tied together. Also, the present invention is applicable to any type of server, for example, web server, LAN server, WAN, intranet server, etc.

An example of the host 102 is illustrated in FIG. 8. The host 102 includes one or more processors, such as processor 201. The processor 201 is connected to a communication infrastructure 806, such as a bus or network). Various software implementations are described in terms of this exemplary computer system. After reading this description, it will become apparent to a person skilled in the relevant art how to implement the invention using other computer systems and/or computer architectures.

Host 102 also includes a main memory 808, preferably random access memory (RAM), and may also include a secondary memory 810. The secondary memory 810 may include, for example, a hard disk drive 812 and/or a removable storage drive 814, representing a magnetic tape drive, an optical disk drive, etc. The removable storage drive 814 reads from and/or writes to a removable storage unit 818 in a well known manner. Removable storage unit 818 represents a magnetic tape, optical disk, or other storage medium that is read by and written to by removable storage drive 814. As will be appreciated, the removable storage unit 818 can include a computer usable storage medium having stored therein computer software and/or data.

US 7,461,148 B1

15

In alternative implementations, secondary memory **810** may include other means for allowing computer programs or other instructions to be loaded into computer system **800**. Such means may include, for example, a removable storage unit **822** and an interface **820**. An example of such means may include a removable memory chip (such as an EPROM, or PROM) and associated socket, or other removable storage units **822** and interfaces **820** which allow software and data to be transferred from the removable storage unit **822** to computer system **800**.

Computer system **800** may also include one or more communications interfaces, such as communications interface **824**. Communications interface **824** allows software and data to be transferred between computer system **800** and external devices. Examples of communications interface **824** may include a modem, a network interface (such as an Ethernet card), a communications port, a PCMCIA slot and card, etc. Software and data transferred via communications interface **824** are in the form of signals **828** which may be electronic, electromagnetic, optical or other signals capable of being received by communications interface **824**. These signals **828** are provided to communications interface **824** via a communications path (i.e., channel) **826**. This channel **826** carries signals **828** and may be implemented using wire or cable, fiber optics, an RF link and other communications channels. In an embodiment of the invention, signals **828** comprise data packets sent to processor **201**. Information representing processed packets can also be sent in the form of signals **828** from processor **201** through communications path **826**.

The terms “computer program medium” and “computer usable medium” are used to generally refer to media such as removable storage units **818** and **822**, a hard disk installed in hard disk drive **812**, and signals **828**, which provide software to the host **102**.

Computer programs are stored in main memory **808** and/or secondary memory **810**. Computer programs may also be received via communications interface **824**. Such computer programs, when executed, enable the host **102** to implement the present invention as discussed herein. In particular, the computer programs, when executed, enable the processor **201** to implement the present invention. Where the invention is implemented using software, the software may be stored in a computer program product and loaded into host **102** using removable storage drive **814**, hard drive **812** or communications interface **824**.

While various embodiments of the present invention have been described above, it should be understood that they have been presented by way of example, and not limitation. It will be apparent to persons skilled in the relevant art that various changes in form and detail may be made therein without departing from the spirit and scope of the invention. This is especially true in light of technology and terms within the relevant art(s) that may be later developed. Thus, the present invention should not be limited by any of the above-described exemplary embodiments, but should be defined only in accordance with the following claims and their equivalents.

What is claimed is:

1. A server comprising:

a host running a single operating system (OS) kernel;

a plurality of isolated virtual private servers (VPSs) supported within the operating system kernel and all the VPSs sharing the same single operating system kernel; a user space/kernel space interface that virtualizes the OS kernel and includes a CPU time scheduler that allocates CPU time between threads of the VPSs and ensures that a thread of one VPS does not adversely affect a CPU time allocation of threads of other VPSs;

16

an application available to users of the VPSs; and an application interface that includes system calls, shared memory interfaces and I/O driver control (ioctl)s for giving the users access to the application.

2. The server of claim 1, wherein each VPS has its own set of addresses and its own set of objects.

3. The server of claim 1, wherein the scheduler ensures that a thread of one VPS does not adversely affect a CPU time allocation of threads of other VPSs by setting guaranteed levels of CPU time usage for each user process and/or VPS.

4. The server of claim 3, wherein each object has a unique identifier in a context of the operating system kernel.

5. The server of claim 1, wherein the server includes a capability of compensating a particular VPS in a later time slice for under-use or over-use of the resource by the particular VPS in a current time slice.

6. The server of claim 1, further comprising isolation of server resources for each VPS.

7. The server of claim 1, wherein each VPS cannot affect an object of another VPS.

8. The server of claim 1, wherein each VPS cannot affect an object of the OS kernel.

9. The server of claim 1, wherein each VPS cannot access information about a process running on another VPS.

10. The server of claim 1, wherein each VPS includes: isolation of address space of a user of one VPS from address space of a user on another VPS; and isolation of application failure effects.

11. The server of claim 1, wherein the host includes any of: a virtual memory allocated to each user; a pageable memory used by the OS kernel and by user processes; physical memory used by the user processes; objects and data structures used by the OS kernel; I/O resources; and file space.

12. The server of claim 10, wherein each VPS includes: a plurality of processes and threads servicing corresponding users; a plurality of objects associated with the plurality of processes and threads; a set of unique user IDs corresponding to users of a particular VPS; a unique file space; means for management of the particular VPS; means for management of services offered by the particular VPS to its users; and means for delivery of the services to the users of the particular VPS.

13. The server of claim 1, wherein the server includes any of the following:

a capability of allocating a resource to a designated VPS; a capability of reallocating the resource to a designated VPS;

a capability of allocating the resource to a VPS in current need of resources;

a capability of reallocating the resource to a VPS in current need of resources;

a capability of dynamically reallocating the resource from one VPS to another VPS when this resource is available; and

a capability of dynamically reallocating the resource from one VPS to another VPS when commanded by the OS kernel.

14. The server of claim 6, wherein the server defines time slices, such that the resource is allocated for each time slice.

US 7,461,148 B1

17

15. The server of claim 6, wherein the server defines time slices, such that the resource is reallocated for each time slice from one VPS to any of another VPS, the OS kernel, an application software daemon and a system software daemon.

16. The server of claim 15, wherein the server dynamically partitions and dedicates the resource to the VPSs based on a service level agreement.

17. The server of claim 1, wherein all the VPSs are supported within the same OS kernel.

18. The server of claim 1, wherein some functionality of the VPSs is supported in user mode.

19. The server of claim 1, wherein the server is implemented as an add-on to any of Microsoft Windows NT server, Microsoft Windows 2000 server, and Microsoft Windows Server 2003 server.

20. The server of claim 1, wherein the server is implemented as an add-on to a server based on a Microsoft Windows product.

21. The server of claim 1, wherein the operating system kernel includes at least one process and thread for processing of user requests.

22. A server comprising:

a computer system running a single operating system (OS) kernel;

a plurality of virtual private servers (VPSs) running on the computer system, wherein each VPS functionally appears to a user as a dedicated server, and all the VPSs sharing the same single operating system kernel;

a user space/kernel space interface that virtualizes the OS kernel and includes a CPU time scheduler that allocates CPU time between threads of the VPSs and prevents a thread of one VPS from adversely affecting a CPU time allocation of threads of other VPSs;

a plurality of applications running within the VPSs and available to users of the VPSs; and

an application interface that includes system calls, shared memory interfaces and I/O driver control (ioctl) for giving the users for giving the users access to the application.

23. The server of claim 22, wherein each VPS is isolated from any other VPS.

24. The server of claim 22, wherein each VPS has its own set of addresses.

25. The server of claim 24, wherein each VPS has its own objects.

26. The server of claim 25, wherein each object has a unique identifier in a context of the OS kernel.

27. The server of claim 1, wherein each VPS manages individual user resource limitations.

28. The server of claim 22, wherein each VPS cannot affect an object of another VPS.

29. The server of claim 22, wherein each VPS cannot access information about a process running on another VPS.

30. The server of claim 22, wherein each VPS includes: isolation of a set of addresses of a user of one VPS from addresses of a user of another VPS; isolation of server resources for each VPS; and isolation of application program failure effects.

31. The server of claim 22, wherein the server includes any of the following resources:

a virtual memory allocated to each user; a pageable memory used by the OS kernel and by user processes;

physical memory used by the user processes; objects and data structures used by the OS kernel;

I/O resources;

file space; and

individual user resource limitations.

18

32. The server of claim 22, wherein each VPS includes the following resources:

a plurality of processes and threads servicing corresponding users;

a plurality of objects associated with the plurality of processes and threads;

a set of unique user IDs corresponding to users of a particular VPS;

a unique file space;

means for management of the particular VPS;

means for management of services offered by the particular VPS to its users; and

means for delivery of the services to the users of the particular VPS.

33. The server of claim 22, wherein the server includes any of the following capabilities:

a capability of allocating a server resource to a designated VPS;

a capability of reallocating the server resource to a designated VPS;

a capability of allocating the server resource to a VPS in current need of that resource;

a capability of reallocating the server resource to a VPS in current need of that resource;

a capability of dynamically reallocating the server resource from one VPS to another VPS when that server resource is available;

a capability of dynamically reallocating the server resource from one VPS to another VPS when commanded by the OS kernel; and

a capability of compensating a particular VPS in a later time slice for under-use or over-use of the server resource by the particular VPS in a current time slice.

34. The server of claim 33, wherein the server defines time slices, such that the server resource is allocated for each time slice.

35. The server of claim 22, wherein the server dynamically partitions and dedicates server resources to the VPSs based on a service level agreement.

36. The server of claim 22, wherein all the VPSs are supported within the same OS kernel.

37. The server of claim 22, wherein some functionality of the VPSs is supported in user mode.

38. The server of claim 22, wherein the server is implemented as an add-on to any of Microsoft Windows NT server, Microsoft Windows 2000 server, and Microsoft Windows Server 2003 server.

39. The server of claim 22, wherein the server is implemented as an add-on to a server based on a Microsoft Windows product.

40. The server of claim 22, wherein the operating system includes at least one process and thread for execution of user requests.

41. A server comprising:

a host running a single operating system (OS) kernel;

a plurality of isolated virtual private servers (VPSs) running on the host, and all the VPSs sharing the same single operating system kernel;

a user space/kernel space interface that virtualizes the OS kernel and includes a CPU time scheduler that allocates CPU time between threads of the VPSs and prevents a thread of one VPS from adversely affecting a CPU time allocation of threads of other VPSs;

wherein the server includes a capability of dynamically reallocating host resources to a first VPS when a second VPS is under-utilizing its resources; and

US 7,461,148 B1

19

wherein the server includes an application interface that includes system calls, shared memory interfaces, and I/O driver control (ioctl) for giving the users access to the application.

42. The server of claim 41, wherein the server includes a capability of compensating a particular VPS in a later period for under-use or over-use of the host resources by the particular VPS in a current period.

43. A method of managing a server comprising:

defining a virtual private server (VPS) ID corresponding to a VPS, wherein multiple VPSs are running on the server and all the VPSs share a single operating system kernel of the server;

wherein the server includes a user space/kernel space interface that virtualizes the operating system kernel and includes a CPU time scheduler that allocates CPU time between threads of the VPSs and prevents a thread of one VPS from adversely affecting a CPU time allocation of threads of other VPSs, and

wherein the server includes an application interface that includes system calls, shared memory interfaces, and I/O driver control (ioctl) for giving the users access to the application;

receiving a request from a VPS process to create an object; creating an internal operating system (OS) kernel representation of the object;

checking whether such an object already exists in OS kernel storage;

if no such object exists in the OS kernel storage, creating an instance of the object to be associated with the VPS ID; and

if such an object already exists in the OS kernel storage, one of rejecting the request and returning the existing object to the VPS process.

44. The method of claim 43, wherein the VPS process is a user mode process.

45. The method of claim 43, wherein the VPS process is an application running within the VPS.

46. The method of claim 43, further including the step of storing a representation of the newly created instance of the object in the OS kernel storage.

47. The method of claim 46, wherein the storage is an OS kernel cache.

48. The method of claim 46, wherein the storage is a data structure in OS kernel memory.

49. A method of managing server resources comprising:

creating a plurality of isolated virtual private servers (VPSs) running on a host and all the VPSs sharing the same single operating system kernel running on the host; the host including a user space/kernel space interface that virtualizes the OS kernel and includes a CPU time scheduler that allocates CPU time between threads of the VPSs and prevents a thread of one VPS from adversely affecting a CPU time allocation of threads of other VPSs, and

wherein the host includes an application interface that includes system calls shared memory interfaces, and I/O driver control (ioctl) for giving the users access to the application;

allocating host resources to each VPS based on a corresponding service level agreement (SLA); and dynamically changing the host resources available to a particular VPS when such host resources are available.

50. The method of claim 49, wherein the host resources allocated to the particular VPS are available due to underutilization by other VPSs.

20

51. The method of claim 49, wherein the host resources include any of CPU usage, disk space, physical memory, virtual memory and bandwidth.

52. The method of claim 49, further including the step of reserving a particular host resource for the particular VPS.

53. The method of claim 49, wherein the step of dynamically changing the host resources allocates resources that exceed SLA guarantees.

54. The method of claim 49, wherein the step of dynamically changing the host resources is performed when the SLA specifies a soft upper limit on resource allocation.

55. The method of claim 49, further including the step of compensating the particular VPS in one time period for host resource underavailability in an earlier time period.

56. A method of managing a server comprising:

creating structures of a single operating system (OS) kernel that handle isolated virtual private server (VPS) specific objects;

initiating instances of VPSs with corresponding VPS IDs, wherein all the VPS instances are supported within the OS kernel and all the VPSs share the same single operating system kernel;

wherein the host includes a user space/kernel space interface that virtualizes the OS kernel and includes a CPU time scheduler that allocates CPU time between threads of the VPSs and prevents a thread of one VPS from adversely affecting a CPU time allocation of threads of other VPSs, and

wherein the host includes an application interface that includes system calls, shared memory interfaces, and I/O driver control (ioctl) for giving the users access to the application;

generating instances of OS kernel objects corresponding to the VPS IDs; and

providing services to users of the VPSs.

57. The method of claim 56, further including the step of allocating resources to the VPSs.

58. The method of claim 57, further including the step of dynamically adjusting the allocated resources for each VPS.

59. The method of claim 56, further including the step of terminating a VPS and all its associated objects and processes based on the VPS ID.

60. A system for managing a server comprising:

means for defining a virtual private server (VPS) ID, wherein multiple VPSs are running on the server and all the VPSs share a single operating system kernel of the server;

wherein the server includes a user space/kernel space interface that virtualizes the operating system kernel and includes a CPU time scheduler that allocates CPU time between threads of the VPSs and prevents a thread of one VPS from adversely affecting a CPU time allocation of threads of other VPSs, and

wherein the host includes an application interface that includes system calls, shared memory interfaces, and I/O driver control (ioctl) for giving the users access to the application;

means for receiving a request from a VPS process to create an object;

means for creating an internal operating system (OS) kernel representation of the object;

means for checking whether such an object already exists in OS kernel storage;

means for creating an instance of the object to be associated with the VPS ID if no such object exists in the OS kernel storage; and

US 7,461,148 B1

21

means for rejecting the request if such an object already exists in the OS kernel storage.

61. The system of claim 60, wherein the VPS process is a user mode process.

62. The system of claim 60, wherein each VPS appears to a user as functionally equivalent to a remotely accessible server.

63. The system of claim 60, wherein the VPS process is an application running within the VPS.

64. The system of claim 60, further including means for storing a representation of the object in the OS kernel storage.

65. The system of claim 60, wherein the storage is a data structure in OS kernel memory.

66. A system for managing server resources comprising:
means for creating a plurality of isolated virtual private servers (VPSs) running on a host, wherein multiple VPSs are running on the host and all the VPSs share a single operating system kernel of the server;

wherein the host includes a user space/kernel space interface that virtualizes the operating system kernel and includes a CPU time scheduler that allocates CPU time between threads of the VPSs and prevents a thread of one VPS from adversely affecting a CPU time allocation of threads of other VPSs, and

wherein the host includes an application interface that includes system calls, shared memory interfaces, and I/O driver control (ioctl) for giving the users access to the application;

means for allocating host resources to each VPS based on its service level agreement (SLA); and

means for dynamically changing the host resources available to a particular VPS when such host resources are available.

67. The system of claim 66, wherein the host resources allocated to the particular VPS are available due to underutilization by other VPSs.

68. The system of claim 66, wherein the host resources include any of CPU usage, disk space, physical memory, virtual memory and bandwidth.

69. The system of claim 66, further including means for reserving a particular host resource for the particular VPS.

70. The system of claim 66, wherein the means for dynamically changing the host resources allocates resources that exceed SLA guarantees to the particular VPS.

71. The system of claim 70, wherein the means for dynamically changing the host resources increases the host resources when the SLA specifies a soft upper limit on resource allocation.

72. The system of claim 66, further including means for compensating the particular VPS in one time period for host resource underavailability in an earlier time period.

73. The system of claim 66, wherein each VPS appears to a user as functionally equivalent to a remotely accessible server.

74. A system for managing a server comprising:
means for creating structures of a single operating system (OS) kernel that handle virtual private server (VPS) specific objects;

means for initiating instances of VPSs with corresponding IDs, wherein all the instances are supported within the OS kernel, wherein all the VPSs share a single operating system kernel of the server;

wherein the server includes a user space/kernel space interface that virtualizes the operating system kernel and includes a CPU time scheduler that allocates CPU time

22

between threads of the VPSs and prevents a thread of one VPS from adversely affecting a CPU time allocation of threads of other VPSs, and

wherein the server includes an application interface that includes system calls, shared memory interfaces, and I/O driver control (ioctl) for giving the users access to the application;

means for generating instances of OS kernel objects corresponding to the VPS IDs; and

means for providing services to users of the VPSs.

75. The system of claim 74, further including means for dynamically adjusting resources allocated to each VPS.

76. The system of claim 74, further including means for terminating a VPS and all its associated objects and processes based on the VPS ID.

77. The system of claim 74, wherein each VPS appears to a user as functionally equivalent to a remotely accessible server.

78. A computer program product for managing a server, the computer program product comprising a computer useable storage medium having computer program logic recorded thereon for controlling a processor, the computer program logic comprising:

computer program code means for defining a virtual private server (VPS) ID, wherein multiple VPSs are running on the server and all the VPSs share a single operating system kernel of the server;

wherein the server includes a user space/kernel space interface that virtualizes the operating system kernel and includes a CPU time scheduler that allocates CPU time between threads of the VPSs and prevents a thread of one VPS from adversely affecting a CPU time allocation of threads of other VPSs, and

wherein the server includes an application interface that includes system calls, shared memory interfaces, and I/O driver control (ioctl) for giving the users access to the application;

computer program code means for receiving a request from a VPS process to create an object;

computer program code means for creating an internal operating system (OS) kernel representation of the object;

computer program code means for checking whether such an object already exists in OS kernel storage;

computer program code means for creating a new instance of the object to be associated with the VPS ID if no such object exists in the OS kernel storage; and

computer program code means for rejecting the request if such an object already exists in the OS kernel storage.

79. A computer program product for managing server resources, the computer program product comprising a computer useable storage medium having computer program logic recorded thereon for controlling a processor, the computer program logic comprising:

computer program code means for creating a plurality of isolated virtual private servers (VPSs) running on a host, wherein multiple VPSs are running on the host and all the VPSs share a single operating system kernel of the server;

wherein the host includes a user space/kernel space interface that virtualizes the operating system kernel and includes a CPU time scheduler that allocates CPU time between threads of the VPSs and prevents a thread of one VPS from adversely affecting a CPU time allocation of threads of other VPSs, and

US 7,461,148 B1

23

wherein the host includes an application interface that includes system calls shared memory interfaces, and I/O driver control (ioctl) for giving the users access to the application;

computer program code means for allocating host resources to each VPS based on a corresponding service level agreement; and

computer program code means for dynamically changing host resources available to a particular VPS when such host resources are available.

80. A computer program product for managing a server, the computer program product comprising a computer useable storage medium having computer program logic recorded thereon for controlling a processor, the computer program logic comprising:

computer program code means for creating operating system (OS) kernel structures that handle virtual private server (VPS) specific objects, wherein multiple VPSs are running on the host and all the VPSs share a single operating system kernel of the server;

24

wherein the host includes a user space/kernel space interface that virtualizes the operating system kernel and includes a CPU time scheduler that allocates CPU time between threads of the VPSs and prevents a thread of one VPS from adversely affecting a CPU time allocation of threads of other VPSs, and

wherein the host includes an application interface that includes system calls, shared memory interfaces and I/O driver control (ioctl) for giving the users access to the application;

computer program code means for initiating instances of VPSs with corresponding IDs, wherein all the instances are supported within the OS kernel;

computer program code means for generating instances of OS kernel objects corresponding to the VPS IDs; and

computer program code means for providing services to users of the VPSs.

* * * * *

EXHIBIT D



US010795659B1

(12) **United States Patent**
Kinsburiski et al.

(10) **Patent No.:** **US 10,795,659 B1**
(45) **Date of Patent:** **Oct. 6, 2020**

(54) **SYSTEM AND METHOD FOR LIVE
PATCHING PROCESSES IN USER SPACE**

(71) Applicant: **Virtuozzo International GmbH,**
Schaffhausen (CH)

(72) Inventors: **Stanislav Kinsburiski,** Moscow (RU);
Alexey Kobets, Seattle, WA (US);
Eugene Kolomeetz, Moscow (RU)

(73) Assignee: **Virtuozzo International GmbH,**
Schaffhausen (CH)

(*) Notice: Subject to any disclaimer, the term of this
patent is extended or adjusted under 35
U.S.C. 154(b) by 0 days.

(21) Appl. No.: **16/178,068**

(22) Filed: **Nov. 1, 2018**

Related U.S. Application Data

(60) Provisional application No. 62/580,611, filed on Nov.
2, 2017.

(51) **Int. Cl.**
G06F 8/65 (2018.01)

(52) **U.S. Cl.**
CPC **G06F 8/65** (2013.01)

(58) **Field of Classification Search**
CPC **G06F 8/65**
See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

9,164,754 B1 * 10/2015 Pohlack G06F 8/658
9,594,549 B2 * 3/2017 Hey G06F 8/60
9,800,603 B1 * 10/2017 Sidagni H04L 63/1433

10,248,409 B1 * 4/2019 Pohlack G06F 8/65
10,657,262 B1 * 5/2020 Cui G06F 21/54
2003/0033599 A1 * 2/2003 Rajaram H04M 1/24
717/173
2008/0101659 A1 * 5/2008 Hammoud G06K 9/00597
382/118

(Continued)

FOREIGN PATENT DOCUMENTS

CN 102037444 B * 7/2014 G06F 8/658
CN 110633090 A * 12/2019

OTHER PUBLICATIONS

S"uBkraut et al. "Automatically Finding and Patching Bad Error
Handling", 2006, IEEE (Year: 2006).*

(Continued)

Primary Examiner — Wei Y Zhen

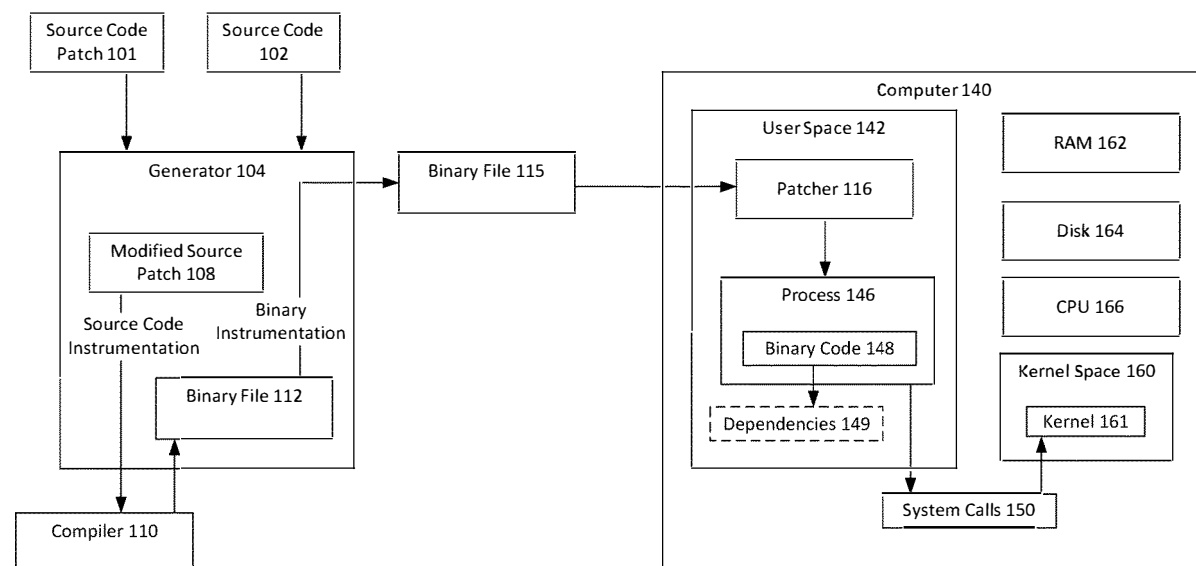
Assistant Examiner — Junchun Wu

(74) *Attorney, Agent, or Firm* — Arent Fox LLP; Michael
Fainberg

(57) **ABSTRACT**

A system and method for live patching a process in user-space is disclosed. In one exemplary aspect, a system for live patching comprises a process executing in userspace in an operating system executed by a hardware processor and a patcher configured to: suspend execution of the process, wherein a memory address space of the process contains binary code executed in the process, and wherein the binary code comprises one or more symbols, map a binary patch to the memory address space of the process, wherein the binary patch contains amendments to the binary code, wherein the binary patch references a portion of the one or more symbols, and wherein the binary patch contains metadata indicating offsets of the portion of the one or more symbols, resolve the portion of the one or more symbols using the offsets in the metadata and resume execution of the process.

22 Claims, 8 Drawing Sheets



US 10,795,659 B1

Page 2

(56) References Cited

U.S. PATENT DOCUMENTS

2013/0104119 A1* 4/2013 Matsuo G06F 8/65
717/173
2016/0170745 A1* 6/2016 Best G06F 8/658
717/122
2016/0188630 A1* 6/2016 Pfeifle G01C 21/32
717/169
2017/0039056 A1* 2/2017 Ninos G06F 3/0653
2017/0168804 A1* 6/2017 Lang G06F 16/128
2017/0308770 A1* 10/2017 Jetley G06K 9/4671

OTHER PUBLICATIONS

Chen et al., “Adaptive Android Kernel Live Patching”, Aug. 2017,
USENIX Association (Year: 2017).*
Xin et al., “Leveraging Syntax-Related Code for Automated Pro-
gram Repair”, 2017, IEEE (Year: 2017).*
Klein et al., “Parallel Tracking and Mapping for Small AR Workspaces”,
2007, IEEE (Year: 2007).*

* cited by examiner

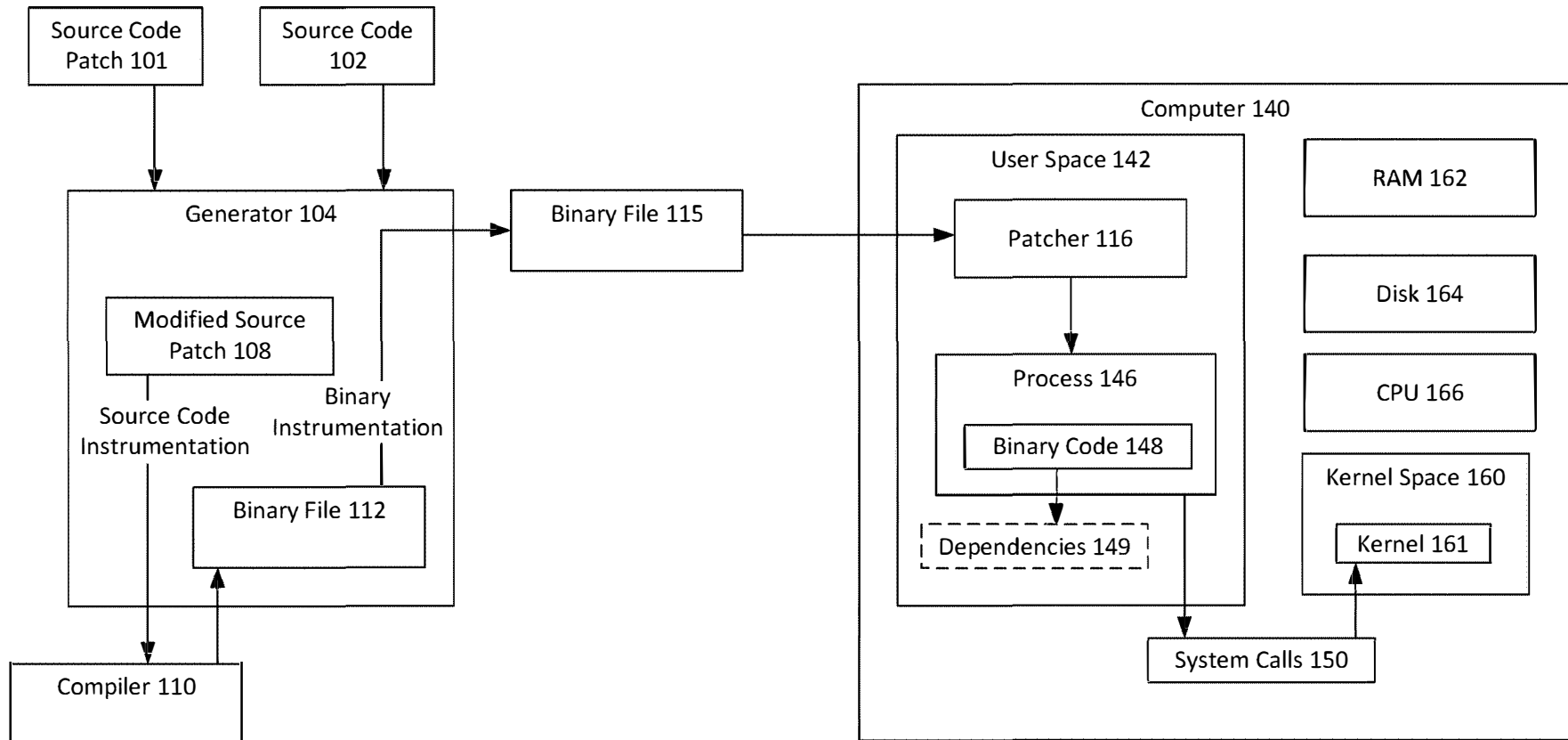


Figure 1

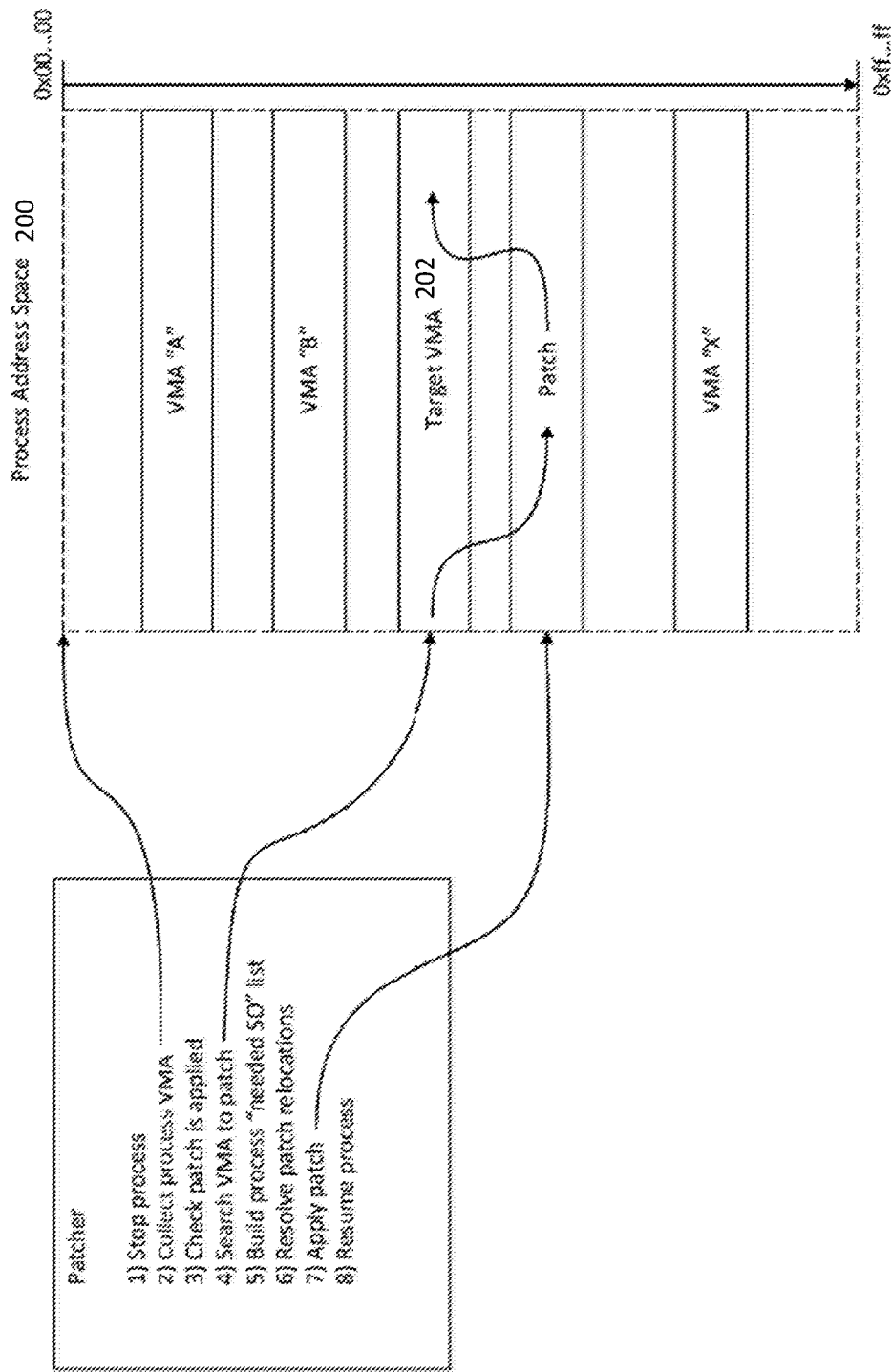


Figure 2

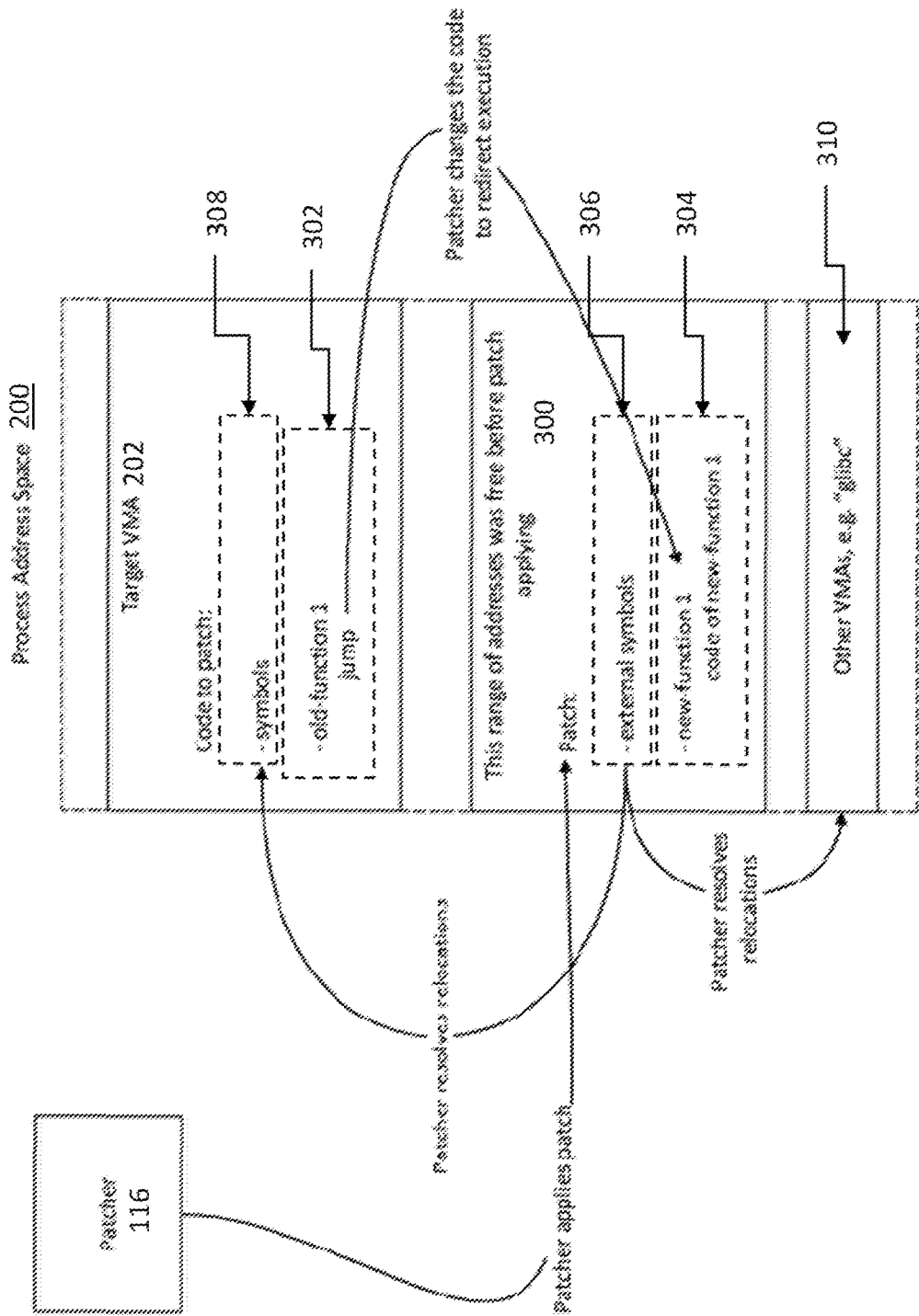


Figure 3

U.S. Patent

Oct. 6, 2020

Sheet 4 of 8

US 10,795,659 B1

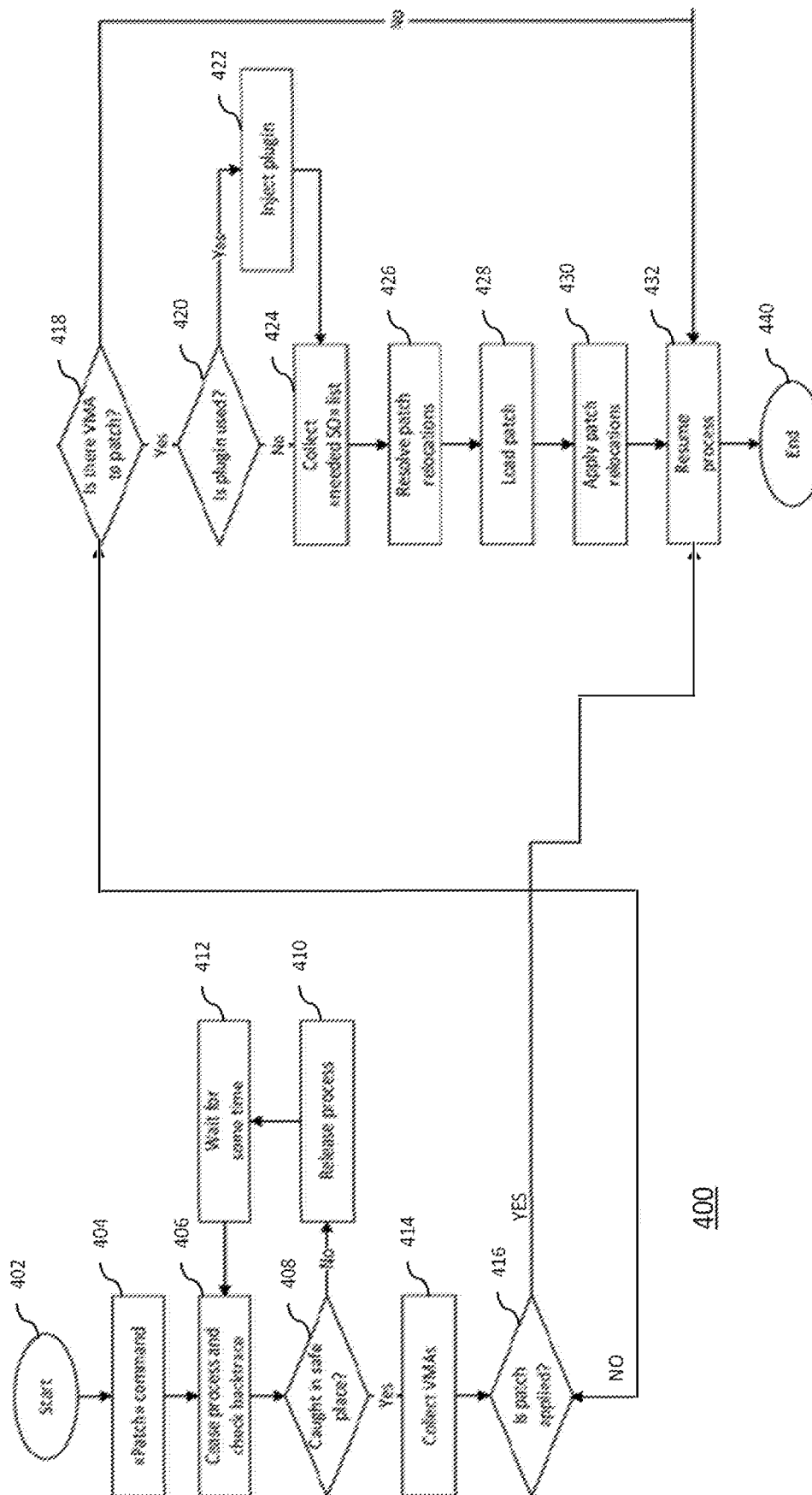


Figure 4

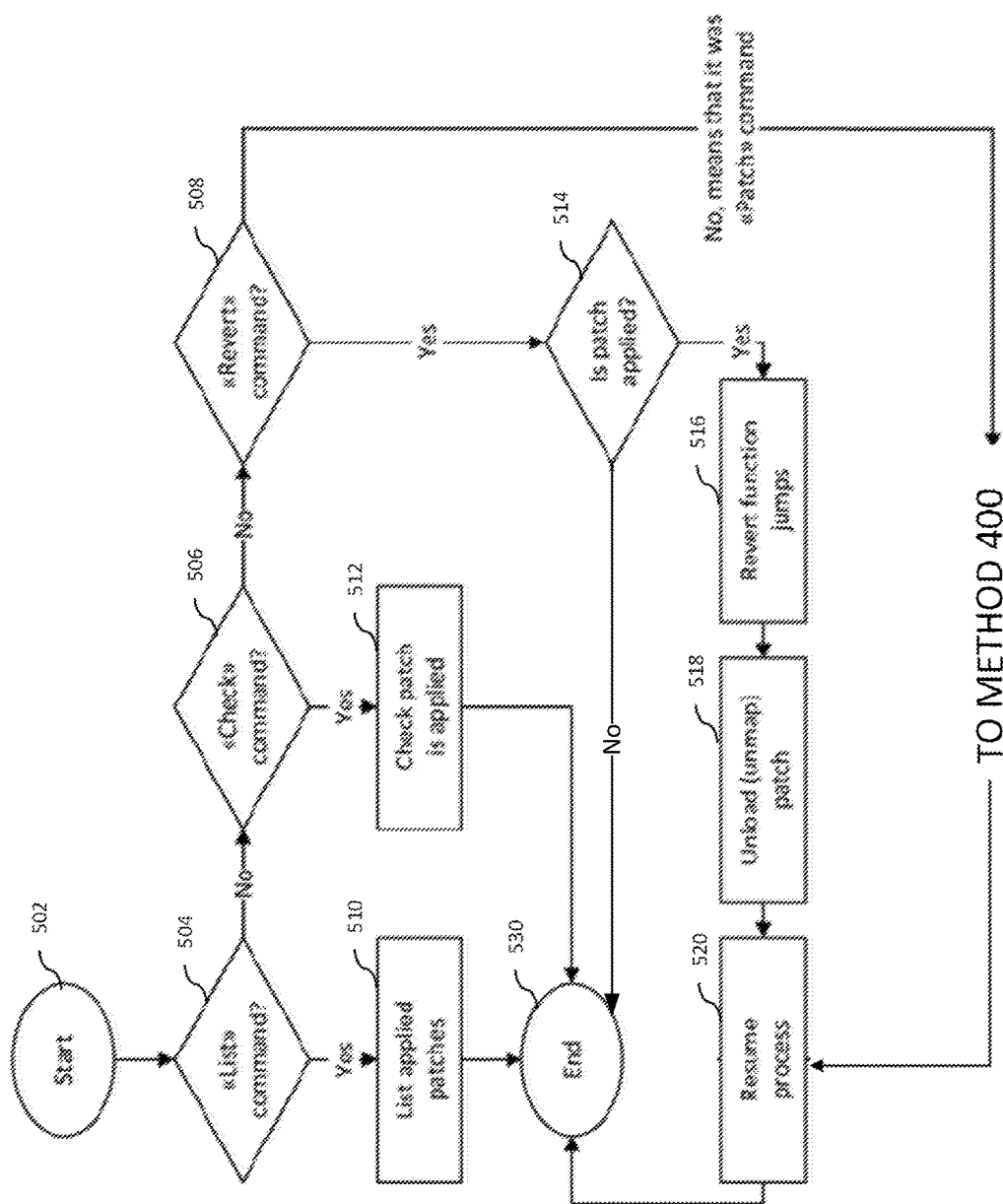


Figure 5

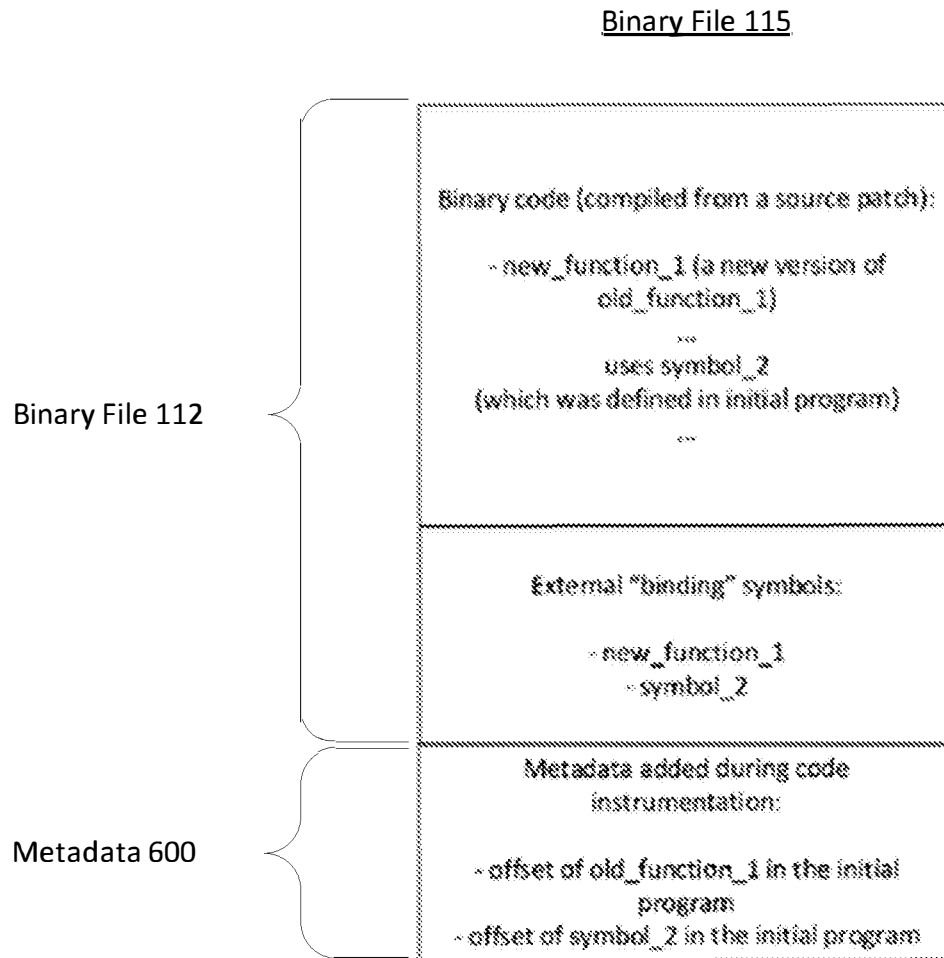
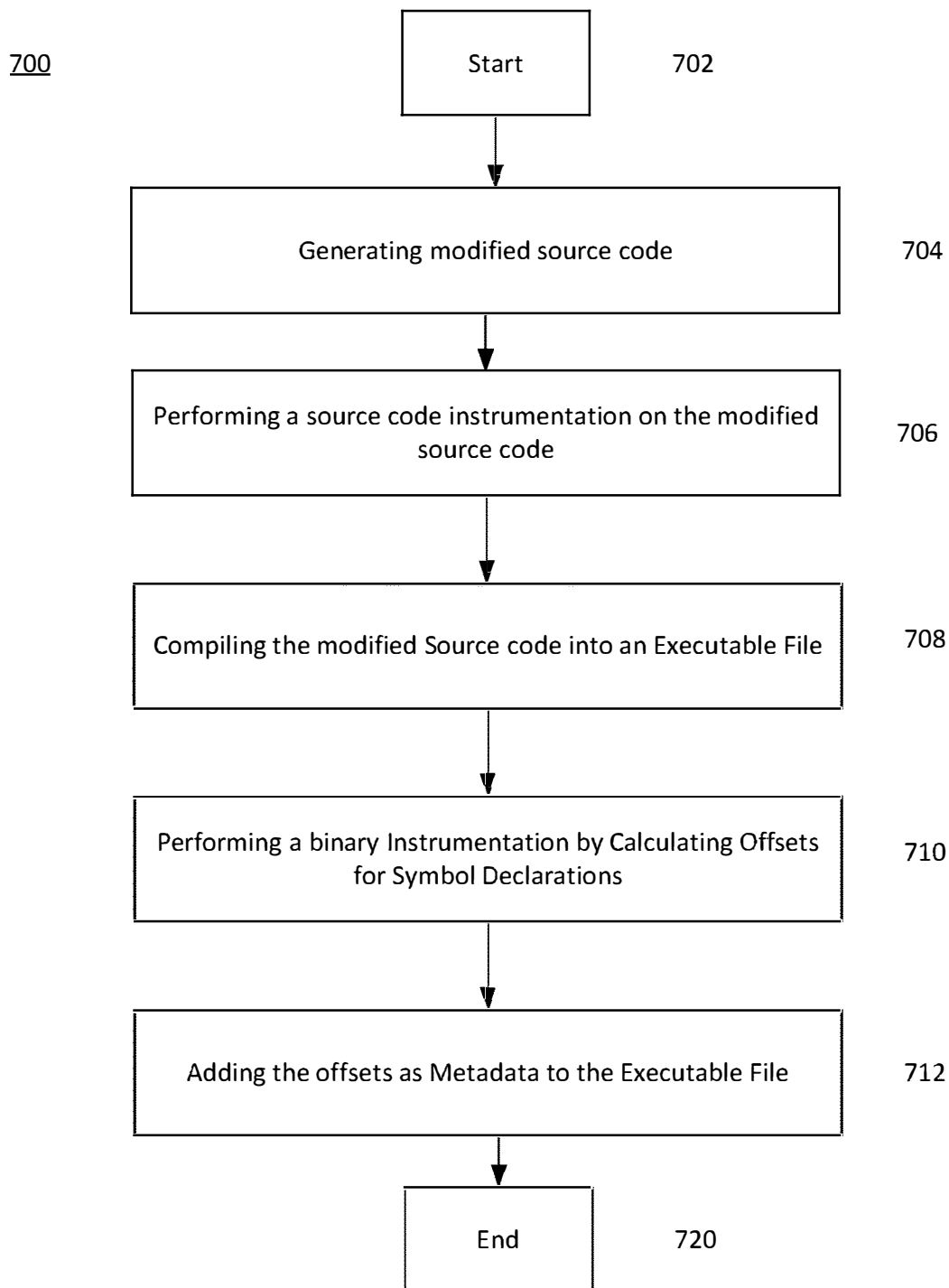


Figure 6

U.S. Patent**Oct. 6, 2020****Sheet 7 of 8****US 10,795,659 B1****Figure 7**

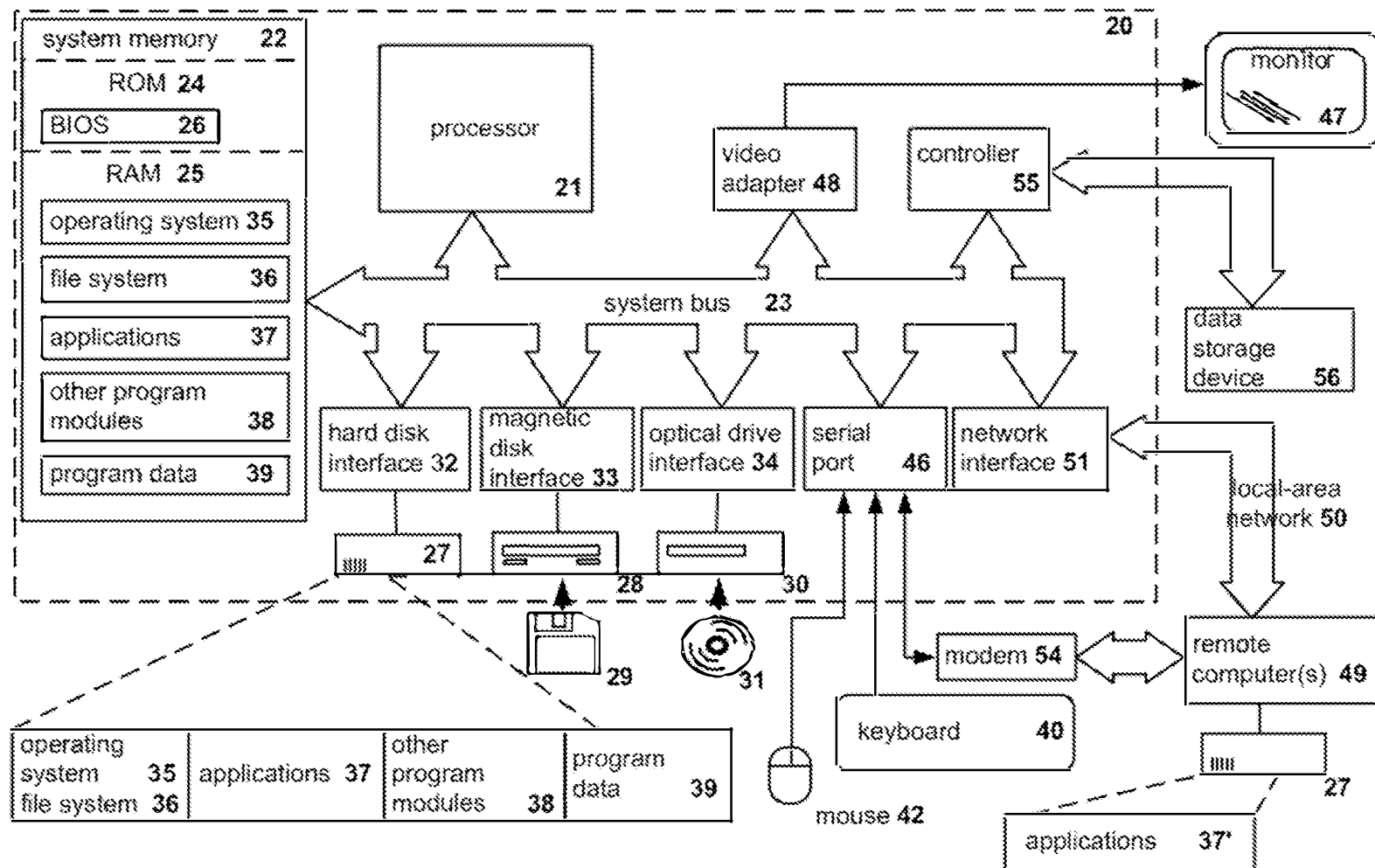


Figure 8

US 10,795,659 B1

1

**SYSTEM AND METHOD FOR LIVE
PATCHING PROCESSES IN USER SPACE****CROSS-REFERENCE TO RELATED
APPLICATIONS**

This application claims the benefit of U.S. Provisional Patent Application Ser. No. 62/580,611 filed Nov. 2, 2017 entitled "System and Method for Userspace Live Patching", which is herein incorporated by reference in its entirety.

FIELD OF TECHNOLOGY

The present disclosure relates generally to the field of applying software updates and, more specifically, to systems and methods for live patching processes in user space.

BACKGROUND

Many software applications run continuously on servers and are generally not stopped, or killed, unless maintenance is required. For example, services such as web servers, application servers and databases are continuously waiting for requests in order to support websites, applications, services and the like. Shutting the software applications down (e.g., killing their executing processes), even for maintenance, is generally undesirable as there may be external services relying on these applications running continuously. In some instances, critical updates, such as security updates, must be applied immediately, or as soon as possible. This leads to administrators waiting until low usage periods of time when the application can be shut down for patching, or a live software patch can be applied.

However, existing solutions for patching running processes, referred to as live patching, operate at the kernel level and can only change running processes in kernel space. These solutions do not run in user space (referring to memory address space outside of kernel space). The available solutions are also functionally limited because they are bound to only a single CPU architecture as they rely on comparisons of binary (or assembler) code of two versions of the program (the old and the new version). The task of comparing binary (or assembler) code is complex and differs entirely across different computer architectures, therefore the various solutions are generally bound to a single architecture. Further, for these solutions to create patch code, the old code of the application being updated and the new code must be compiled in the same way, in some cases with precisely the same compiler options and the same compiler version. This makes developing live patch code increasingly difficult to perform, maintain and modify, and even still does not work in user space.

Therefore, there is a need in the art for a system and method for live patching processes in user space that is architecturally agnostic and is easy to develop and maintain.

SUMMARY

A system and method is disclosed herein for live patching processes in userspace. In an exemplary aspect, the disclosure provides a system for applying a patch to a running process in userspace. The system may comprise a process executing in userspace in an operating system executed by a hardware processor, a patcher configured to suspend execution of the process, wherein a memory address space of the process contains binary code executed in the process, and wherein the binary code comprises one or more sym-

2

bols, map a binary patch to the memory address space of the process, wherein the binary patch contains amendments to the binary code, wherein the binary patch references a portion of the one or more symbols, and wherein the binary patch contains metadata indicating offsets of the portion of the one or more symbols, resolve the portion of the one or more symbols using the offsets in the metadata and resume execution of the process.

In one aspect, the patcher is configured to map the binary patch by injecting parasite code into the process, transferring control to the parasite code, mapping, using the parasite code, the binary patch to the memory address space by executing instructions in the parasite code.

In another aspect, the patcher is configured to resolve symbols by finding an address of at least one symbol referenced by the binary patch, in the binary code executed in the process and linking the at least one symbol by writing the address that was found into a specified location in the binary patch that is mapped into the memory address space of the process.

In another aspect, the patcher is configured to find the address by calculating the address of the at least one symbol by adding an offset taken from metadata stored in the binary patch to the address of a beginning of the binary code executed in the process in the memory address space of the process.

In another aspect, the patcher is further configured to determine when there is no metadata information in the binary patch associated with the one or more symbols and import the one or more symbols from a library, wherein the library is mapped to the memory address space of the process, when the library is not yet mapped.

In another aspect, the patcher is further configured to amend binary information of the process to allow control transfer from instructions contained in the executing process prior to the patch, to instructions contained in the binary patch.

In another aspect, the patcher is further configured to resume execution of the process, without mapping the binary patch and without resolving symbols in the binary patch, when the process is executing a portion of the associated binary code that contains symbols that are amended in the binary patch and wait until the patch is applied safely.

In another aspect, the patcher applies the patch when, at the moment when execution of the process was suspended, the process was not executing functions for which the binary patch contains amendments.

In an exemplary aspect, the disclosure provides a method for live patching a process in user space. The method may comprise stopping execution of the process executing in user space on a hardware processor, wherein a memory address space of the process contains binary code executed in the process, mapping a binary patch to the memory address space of the process, wherein the binary patch contains amendments to the executable binary code, resolving symbols used by the binary patch using metadata regarding the symbols, the metadata stored in the binary patch and resuming execution of the process when the patching is complete, or patching has failed.

According to another exemplary aspect, the disclosure provides a non-transitory computer-readable medium storing therein instructions for executing a method for live patching a process in user space. The instructions may comprise stopping execution of the process executing in user space on a processor, wherein a memory address space of the process contains executable binary code executed by the

US 10,795,659 B1

3

process, mapping a binary patch to the memory address space of the process, wherein the binary patch contains amendments to the executable binary code, resolving symbols used by the binary patch using metadata regarding the symbols, the metadata stored in the binary patch and resuming execution of the process when the patching is complete, or patching has failed.

The above simplified summary of example aspects serves to provide a basic understanding of the present disclosure. This summary is not an extensive overview of all contemplated aspects, and is intended to neither identify key or critical elements of all aspects nor delineate the scope of any or all aspects of the present disclosure. Its sole purpose is to present one or more aspects in a simplified form as a prelude to the more detailed description of the disclosure that follows. To the accomplishment of the foregoing, the one or more aspects of the present disclosure include the features described and exemplarily pointed out in the claims.

BRIEF DESCRIPTION OF THE DRAWINGS

The accompanying drawings, which are incorporated into and constitute a part of this specification, illustrate one or more example aspects of the present disclosure and, together with the detailed description, serve to explain their principles and implementations.

FIG. 1 illustrates a block diagram of a system for live patching processes in accordance with aspects of the present disclosure.

FIG. 2 illustrates the address space of a process being patched, in accordance with aspects of the present disclosure.

FIG. 3 illustrates the patch being applied in the process address space, and existing code being redirected to the patch code.

FIG. 4 is a flow diagram of a method for live-patching an executing process in user space in accordance with one aspect of the disclosure.

FIG. 5 is a flow diagram illustrating a method patching the process in accordance with another aspect of the disclosure.

FIG. 6 is a block diagram illustrating portions of the executable file in accordance with an aspect of the disclosure.

FIG. 7 is a flow diagram for a method for generating a binary patch in accordance with an aspect of the disclosure.

FIG. 8 illustrates a block diagram of a general-purpose computer system on which the disclosed system and method can be implemented according to an exemplary aspect.

DETAILED DESCRIPTION

Example aspects are described herein in the context of a system, method and computer program product for live patching processes in user space. Those of ordinary skill in the art will realize that the following description is illustrative only and is not intended to be in any way limiting. Other aspects will readily suggest themselves to those skilled in the art having the benefit of this disclosure. Reference will now be made in detail to implementations of the example aspects as illustrated in the accompanying drawings. The same reference indicators will be used to the extent possible throughout the drawings and the following description to refer to the same or like items.

FIG. 1 illustrates a block diagram of a system 100 for live patching processes in accordance with aspects of the present disclosure.

4

The system 100 may comprise a generator 104, a patcher 116 and a computer 140. According to some aspects, the system 100 may only include patcher 116 and the computer 140. In one aspect, the generator 104 may be executed on a different computer system than computer 140 while in other aspects, the generator 104 may execute on computer 140. In one aspect, the patcher 116 and the generator 104 may execute on entirely different computers than each other, patcher 116 executing on computer 140. In another aspect, the patcher 116 may comprise a plurality of components, whereas at least one component or process of the patcher 116 is executed on computer 140. In some aspects, the system 100 may be comprised of the patcher 116 executing at least one process on the computer 140, the generator 104, or both.

The system 100 (which contains at least the components of the patcher 116 components working on the computer 140) patches an application (i.e., process 146) that is running on computer 140 using a patch (i.e., a binary file 115) generated by generator 104 and then applies the patch using patcher 116. The patcher 116 may integrate the binary file 115 into address space of the process 146, thereby patching the process 146 without killing the process 146. Accordingly, the amount of downtime for the computer 140 is minimized or eliminated entirely because the patch is performed “live”. The binary file 115 is a binary file that contains compiled code and data used by the code. According to exemplary aspects of the present disclosure, the binary file 115 can be in any format, depending on the OS in which the process 146 runs. For example, the binary file 115 can be a Portable Executable (PE) file, an Executable and Linkable Format (ELF) file, Mach-O file, or the like. In some aspects, the binary file 115 is a dynamically loaded library (or a shared object), i.e. a file containing executable binary code, that can be loaded to a process and one or more portions of the binary file 115 can be executed in the process.

In one aspect, the computer 140 may contain system memory divided into two regions: user space 142 and kernel space 160. The kernel space 160 is where the kernel of the operating system executes and provides kernel-level services such as controlling the RAM 162 and the disk 164. The user space 142 is the set of memory locations in which user processes run, i.e., the memory not occupied by the kernel space. The user space 142 is generally where user applications and processes are executed, or the address space that is not the kernel, or processes running in non-privileged mode.

In some aspects of the disclosure, the computer may execute a process 146, which may access the kernel space 160 by making a system call 150. At the beginning, the process 146 (corresponding to the application that is being patched) is executing binary code 148 of the initial (unpatched) version of an application, or in other words the computer 140 executes an executable file of the non-patched version of the application.

In other aspects of the disclosure, the process 146 may be executed within a container or a virtual machine. The container is an isolated execution environment for processes. The operating system of the computer 140 may comprise a plurality of containers.

In some aspects, the generator 104 generates the binary file 115 (alternatively referred to as the patch throughout the disclosure). In some other aspects, the system 100 does not contain a generator and the patcher 116 works with a binary patch (e.g., binary file 115), which was generated like following, or which has the similar characteristics as the binary file generated in a way described below. A source code patch 101 is provided to the generator 104. The source

US 10,795,659 B1

5

code patch **101** may be a set of files that contain source code identifying a difference between an unpatched version of the application and a patched version of the application that is executing in process **146**. The source code patch **101** may be generally written using a high-level compiled programming language such as C or C++, though there is no limitation to the language that is contemplated in this disclosure. In one aspect, the source code patch may be a “patch series” as used in LINUX.

According to one aspect, the generator **104** may also be provided source code **102** of the unpatched application currently executing in process **146** on computer **140**. In another aspect, the generator **104** may be provided the unpatched version of the application source code and the patched versions of the application source code. Subsequently, the generator **104** creates the source code patch **101**.

According to an exemplary aspect, the generator **104** determines that particular functions are modified or particular variables are accessed by functions in the source code patch **101**. The generator **104** then may move all effected functions to a new file, the modified source patch **108**. In some aspects, the modified source patch **108** or even binary file **112** can be input to the generator **104** instead of **101** and **102**. However, this modified source patch **108** cannot be compiled yet because there may be references to symbols (i.e., variables, functions, etc.), which are not defined in the modified source patch **108**.

Accordingly, the generator **104** further may perform a source code instrumentation (first instrumentation) by modifying the modified source patch **108**. The first instrumentation comprises marking a portion of the symbols used in the modified source patch **108**, but not defined in the modified source patch **108**, as being defined in a location outside of the modified source patch **108**, for example by marking the symbols as “global” and/or “external”. The marking of a portion of the symbols may be performed for those symbols that were defined in the source code **102** and for symbols that are imported from library dependencies. In one aspect, marks (e.g., like “global” and “external”) also indicate that the symbols are declared outside of any particular block (e.g., any particular function) so they are available to several functions in the software. In some aspects, macros can be used to perform the source patch instrumentation.

The generator **104** may then invoke compiler **110** (or a similar component able to create binary code) to compile the modified source patch **108** (with source code instrumentation) to generate a binary file **112** (e.g., a shared object file).

After the binary file **112** is generated (or alternatively, when generator **104** receives binary file **112**), the generator **104** performs a binary instrumentation (second instrumentation) step to include metadata (also referred to as binary instrumentation) to the binary file **112**, thereby producing binary file **115**, wherein the binary file **115** is alternatively referred to as a binary patch file or simply a patch, as shown in FIG. 6. According to one aspect, the instrumentation information comprises metadata that contains offsets of a portion of the symbols identified in the first instrumentation step by the generator **104**, i.e., “binding” symbols which are symbols from the original application also used in the patch. In one aspect, the metadata is written to a new section of the binary file **112**. In another aspect, the metadata is written to an existing section of the binary file **112**, (e.g., to the Global Offset Table of the shared object, to Import or Relocation section of a PE file, or the like).

In one aspect, the offset of the symbol is constant and does not depend on how the binary file (e.g., binary file corresponding to the binary code **148**) has been loaded to the

6

process address space and in which address range. The offset of a symbol can be calculated from any predetermined place in the code. For example, if offsets are calculated from a base load address of the binary (e.g., binary code **148**), then the real address of a symbol will be the base load address of the binary code plus the offset. For example, if the offsets are calculated from the first address in the target VMA, then the real address of a symbol will be the first address in target VMA plus the offset. The offsets value depend on how the binary has been built (by a compiler or linker), so the offsets can be computed without loading the binary to any process.

In some aspects, the metadata that will be included in the binary file **115** also contains metadata about locations (in the old version of binary code) of the functions that should be replaced by corresponding functions of the patch. In other words, metadata may contain offsets to the old versions of functions in the old (un-patched) binary code.

In some aspects, the generator **104** uses the binary code of the application (that should be patched) to create the metadata (i.e., to calculate offsets). In other aspects, the generator **104** uses debug information (e.g., debugging symbols, or any kind of information provided by the compiler for the debugger) corresponding to the binary code of the application (that should be patched) to create the metadata (i.e., to get offsets). Usually debug information contain at least some of the offsets that are needed to create the metadata. Sanity checks can be performed using the debug information, discussed further below.

In some aspects, the generator **104** may perform sanity checks, comprising checking the types and sizes of symbols in the source code patch **101** and the binary file **112** to determine whether the source code instrumentation was performed properly. In one aspect, the generator **104** may analyze whether the instrumentation was performed correctly by, for example, verifying types and sizes of symbols. This helps to avoid errors in compilation and also to bind contexts of the process **146** and the later-generated binary file **115**, discussed below.

In this manner, the generator **104** is platform agnostic, i.e., the generator **104** is independent of the architecture of computer **140**. The generator **104** can create patches for different architectures, e.g., x86 based systems or x64, arm, MIPS, S390, PPC64, aarch64 systems and the like. The generator **104** may, in one aspect, generate a patch for the same program or application for different platforms, given some options of the generator **104** and the compiler **110**.

In some aspects, the generator **104** does not create binary file **112**, but merely includes service information (i.e., metadata) with the binary file **112** needed to apply the patch.

Now, patcher **116** may patch the application executing in process **146** using binary file **115**. In one aspect, the patcher **116** may execute in user space **142**. In another aspect, the patcher **116** may comprise various components where a portion of the components of the patcher **116** execute on a different computer than computer **140**, over a network, or the like, but at least one of the components is executed in a process executing on computer **140**. In this aspect, a plugin may be used to create a socket on the computer **140** using a library and to send information or commands to the patcher **116** executing in a process on the computer **140** from a portion of the components of the patcher **116** located remotely. The plugin (or a component of the patcher working on the computer **140**) is able to map and un-map a patch or may even send a list of shared objects needed for executing the code from the patch to the computer **140**. The patcher **116** may comprise an applier component and a

US 10,795,659 B1

7

symbol resolver component and may be a program or a set of programs that applies the patch to the application.

According to one aspect, the patcher 116 may first stop the execution of process 146 as shown in FIG. 2. The process address space 200 may comprise many different virtual memory areas (VMAs) such as “A”, “B”, and “X”. Each VMA is a contiguous range of addresses that the process can use. The VMAs are areas in the process address space 200 where, for example, executable code or data is loaded. E.g., a VMA may correspond to a file mapping containing binary code 148 of the process. In this disclosure, binary code 148 implies executable code executing in the process and associated data. The target VMA to be patched (i.e., an address range which contains the code that should be changed/replaced by code from the binary patch) is, in this example, determined to be target VMA 202. The patcher 116 temporarily stops the process 146 and collects the VMA (or any of its analogs, depending on the OS in which the process 146 runs) for the process 146. In one aspect of the disclosure, stopping the process is achieved using the “ptrace” interface, which, in some aspects, sends a SIGTRAP signal to the process 146. The patcher 116 determines whether the patch has been applied, and if not, searches for and finds the target VMA 202. In one aspect, determining whether a patch has been applied comprises determining which patches have already been applied (e.g., the patcher can store information about applied patches), or searching and analyzing the virtual address space of the process 146 to find the functions that were patched or may need patching. The patcher 116 then builds a list of needed shared objects, e.g., to identify symbols exported by which shared objects should be resolved in the patch.

In this aspect, the patcher 116 resolves the symbols in the patch by identifying where the binary code 148 has been loaded in the address space of the process 146 (e.g., by identifying the base load address of binary code 148, or the first address in the target VMA 202, depending on implementation). The patcher 116 adds the offset (for each respective symbol) stored in the metadata of binary file 115 to the identified address to obtain the address of each symbol.

The patcher 116 finds free space in the address space 200 of process 146 and injects parasite code to address space 200 (e.g., writes a set of instructions belonging to parasite code, writes a set of instructions that map parasite code, and transfers control to the written instructions, or any other suitable method). The parasite code is binary code that is injected into process 146, and in turn, the binary code injects code of the patch by loading one or more portions of the binary code from the binary file 115 into free space of the process address space 200. In one aspect, the parasite code maps one or more portions (or the entire contents) of the binary file 115 into the address space 200 or reading contents of the binary file 115 and writing one or more portions of the contents into the address space 200. In one aspect, the one or more portions of the binary file 115 include the compiled functions (e.g., those that were contained in the modified source patch 108 and modified and compiled by the generator 104), while in other aspects the one or more portions also include the metadata (e.g., those created during the second instrumentation step performed by the generator 104). For example, when the metadata information is written to a separate section of the binary file 115, the one or more portions is also loaded. In some aspects, the patcher 116 then modifies the binary code 148 so that the old function of the application, contained in binary code 148, transfers control to the new function, contained in the instructions from the binary file 115. In some aspects, the patcher 116 may modify

8

the old function by inserting jump instruction (or any other instruction responsible for transferring control/changing flow of control; conditional jump, call, enter, etc.) in the old function to jump to the new function. In another aspect, jump instructions may be inserted from the new function back to the old function. In other aspects, the patcher 116 inserts a call instruction to the new function in the old functions. Accordingly, when the old functions are called, control is transferred to the new function. In other aspects, other transfer of control methods are used. At this point, the binary patch has been loaded into the address space 200, symbols have been resolved and control transfers have been made from old functions to new functions. The patcher 116 subsequently resumes the process 146. In one aspect, the process is resumed by the patcher 116 performing a system call to kernel 161 to resume the suspended process 146. At completion, the process 146 is executing a patched version of the application while little to no downtime has been experienced by users of the computer 140. While the code being executed in process 146 on computer 140 is not the same as the code of the new version of the application, the behavior of the code being executed and the new version of the application is equivalent.

The applying of the instructions and/or metadata contained in binary file 115 as a patch to the application executing in process 146 is detailed with respect to FIG. 3. The original binary code 148 of the un-patched application is located in the Target VMA 202. The patcher 116 then links/attaches to the process 146 using any means that provides the ability to attach to the process, or gives read/write access to process’s memory, e.g., a debugging interface. According to one aspect, the patcher may work outside of the process address space 200. According to another aspect, the patcher 116 may be started with special permissions and/or privileges required to read and write to executing processes on computer 140, for example “root” privileges in Unix-based computer systems.

In some aspects, the patcher 116 temporarily stops the process 146 and obtains direct access to the process 146. In one aspect of the disclosure, the patcher 116 may use “ptrace”, debugging interface, “libCompel” library, or any similar interface to access the process 146. In some aspects of the disclosure, “libCompel” may be used to link to the process 146. After the process is temporarily stopped, the patcher 116 maps the binary file 115 to the address space of the process. The mapping is performed by the patcher 116 by loading the binary file 115 into address range 300 that was previously free prior to applying the patch.

In one aspect, the patcher 116 loads the instructions from the binary file 115 by injecting “parasite code” (also referred to as a “binary blob” in this disclosure) to the address space of the process 146. Instructions contained within the parasite code are executed on the computer 140 in the context of the process 146 by transferring execution from instructions in binary code 148 to the parasite code. The instructions in the parasite code load one or more portions of the contents of the binary file 115 into the address space 200. The patch (e.g., the instructions from binary file 115), according to one aspect, may be mapped to the process 146 (e.g., using the “mmap” system call or any of its analogues in any operating system), though other methods may be used. References in the memory mapping of the binary file 115 are then resolved by patcher 116 (e.g., without using a dynamic linker of the operating system; or, in other aspects, with the help of a dynamic linker of the operating system in case of references not referencing to 148 but to other shared objects).

US 10,795,659 B1

9

The mapping of (e.g., special data structures, like relocations, contained in VMA corresponding to the binary file 115) the binary file 115 contains references to unresolved external symbols in section 306 (e.g., functions and variables) that are not defined in the modified source patch 108 (and similarly for binary file 115), but are defined in the original binary code 148 in code portion 308. Therefore, the patcher 116 may search for these external symbols in the binary code 148 and calculate their addresses using metadata in the mapping of the binary file 115. A symbol address for each of these symbols (or in other aspects, symbols having corresponding metadata in the patch) may be determined by adding a symbol offset stored in metadata and the address of the beginning of the Target VMA 202 or the base load address of binary code 148. In one aspect, the patcher 116 also links libraries and dependencies used in the patch, if not already linked to the process 146, and resolves binding symbols in the binary file 112.

The offset of symbols of the portion of code 308 may be stored in the metadata of the binary file 115 generated during the second instrumentation performed by generator 104. Each calculated symbol address is written into section 306 of the file mapping of the binary file 115, or to any appropriate place, for example a special data structure in the mapping, depending on the binary patch file format and the OS. In some aspects, the special data structure may be a global offset table when the binary patch is a shared object. If there is no metadata information in the binary file 115, the symbols are all external already and located in a dependency library, i.e., the symbols are not defined in the original binary code 148. In this case, the patcher 116 behaves similarly to a dynamic linker in an OS.

Usually programs that execute in user space use functions from dynamic libraries. To allow the programs to use dynamic libraries, a dynamic linker of the underlying operating system may resolve dependencies as follows, according to one aspect. A binary file that is executed on a computer, e.g., computer 140, contains a list of needed functions from different libraries. Normally, a dynamic linker finds these libraries and maps the libraries to the process address space (e.g., 200) when needed. While loading the binary of the program, the dynamic linker writes the address of some of the linker's resolver routines instead of the addresses of functions from the libraries that should be used by the binary later. These libraries are referred to as dependencies. Accordingly, when the first attempt to call a function, e.g. "printf", occurs in a binary, the dynamic linker is called instead of the original function. The dynamic linker finds a library containing the function "printf" (in some aspects, according to some priority algorithm) and writes the address of this function in the library to particular place in the mapping of the binary file. In this aspect, a conventional dynamic linker (e.g., an operating system linker) is not used to resolve symbols during patch application because "binding" symbols are internal for the program being patched. An OS dynamic linker will either not find the library containing the called function, or may find an incorrect library, making the patched program behavior unpredictable.

In one aspect, the code of such a binary file may be "position independent code", or PIC, where a real address of a function is not used when calling the function, but the address of a position in the code or data where the real address of the function will be written. While loading a dynamically linked binary, a dynamic linker attempts to find the address of the function and write the address in a portion of memory of the mapping of the binary file. Subsequently,

10

an additional "jump" will occur with every call of the function, making aspects of the present disclosure applicable.

In some aspects, the patcher 116 may calculate the address in the process address space 200 of the "binding" symbol, and may write this address to the section 306 (particular data structure, e.g., global offset table) of the file mapping of binary file 115. In some aspects, the patcher 116 may calculate the real address in the process address space 200 of the static symbol, and may write this address to a global offset table of the binary patch. Accordingly, after applying a patch, any call to a particular external symbol (e.g., variable or function) in the address range 300 will be redirected to the code portion 308 in the original binary code 148.

According to one aspect, the patcher 116 resolves the links to symbols from libraries (in some aspect, for those symbols that do not have corresponding metadata in the patch) while the process 146 is temporarily stopped. In another aspect, the patcher uses the dynamic linker to resolve at least some of such symbols. In yet another aspect, the patcher 116 resolves the links during loading of the binary file 115 into the address range 300 through a dynamic linker, an underlying component used by the OS to resolve dependencies, only when the symbol is actually accessed from the context of the process address space 200.

The patcher 116 amends the binary code 148 to redirect execution of the program from the binary code 148 to the memory mapping of the binary file 115 and then links the binary file 115 with data in the process 146 as described above.

Using the above mentioned methods, the patcher 116 determines which part of the binary code 148 is being executed. Specifically, if the code currently being executed is the code that will be updated by the patch, then the patcher 116 waits until this code section is no longer being used to continue. In some aspects, such a check can be performed anywhere between steps 1-6 shown in FIG. 2.

As for the redirection, the patcher 116 amends the binary information (e.g. binary code 148) to allow redirecting execution (also referred to as transferring control) from the old code function, portion 302, to the new code fragment in portion 304 in the code loaded from binary file 115. The redirection of execution is performed, for example, by adding a "jump" or "call", or a the like, instruction in portion 302 in the very beginning of the old function or where the old function would have been called. The new code portion 304 is written such that execution flow is returned back to caller of the old function after the new function completes in portion 304 of binary code 148.

In some aspects, in order to perform the modification to the instructions in portion 302, the patcher 116 may switch the "context" of the process 146 so the patcher 116 can write to the address space of another process. In one aspect, the modification of the instructions can be performed by code inside the process 146, such as the parasite code that was used to load the binary file 115 to the address space of the process. In another aspect, the modification can be performed by another set of code executed from the context of the process 146, or from (for example) the patcher 116. In the case where the patcher 116 performs the modification, "ptrace" debug interface or the like, may be used that permits writing to any portion of memory of the stopped process.

Patcher 116 then releases the process 146, and the process 146 resumes execution. The process 146 is now live-patched in user space.

US 10,795,659 B1

11

FIG. 4 is a flow diagram of a method 400 for live patching an executing process in user space in accordance with one aspect of the disclosure.

Method 400 is one implementation of the patcher 116 according to exemplary aspects of this disclosure, as executed by processor 21 of computer system 20 shown in FIG. 8.

The method begins at step 402 and proceeds to 404 where patcher 116 receives the patch command.

At step 406, the patcher 116 ceases (e.g., temporarily “stops”) the process to be patched, e.g., process 146.

At step 408, the patcher 116 determines whether the process has been caught in a safe place (for example, by examining the backtrace of the process), e.g., the process is not executing functions that should be replaced by the patch, and does not contain such functions in stack trace. In some aspects, the determination at 408 is performed in order to prevent old versions of functions from execution.

When the process is not stopped in a safe place, the method proceeds to step 410, where the process is released and the patcher 116 waits for a predetermined period of time at step 412 (or waits until such function will end) and returns to step 406. If at 408, the process is stopped at a safe place, the patcher 116 collects the virtual memory addresses of the process at 414.

If the patch has already been applied at 416, the method proceeds to step 432, where the process is resumed and the method ends at step 440.

However, if the patcher 116 determines that the patch has not been applied, the method proceeds to 418.

At 418, the patcher 116 determines whether there is a virtual memory area that contains code to be patched (i.e., a VMA that corresponds to a file containing binary code executed in process). If there is no VMA to patch, the method proceeds to 432 where the process 146 is resumed and the method terminates at 440.

However, if there is a VMA to patch, the method proceeds to another determination 420 to determine if a plugin is used. If a plugin is used, it is injected by patcher 116 at 422. The method then proceeds to 424, whether or not a plugin is used, and collects a list of “needed SO” or needed shared objects or needed dynamic libraries. In some aspects, the list is collected so that patcher 116 may determine which libraries are needed for the patch and may resolve external symbols imported from libraries in the patch.

At 426, the patcher 116 resolves the patch relocations as shown in FIG. 3.

The patcher 116 then loads the patch (e.g., the instructions and/or metadata contained in binary file 115) into the available VMA, e.g., address range 300 from FIG. 3 at 428. According to one aspect, loading the patch entails injecting the parasite code into the process address space 200 shown in FIG. 2, and mapping one or more portions from the binary file 115 into the address range 300 by parasite code.

Patch relocations are applied at 430 by the patcher 116. Patch relocations comprise relocations of any external symbols such as external symbols in section 306 in FIG. 3, to other VMAs or libraries such as “glibc” or the like. If an old method is being patched, jump or call instructions are inserted into the binary code of the process (e.g., binary code 148) to transfer control from the old function to the new function defined in the patch, and then to return to after the old function in portion 302 of FIG. 3.

At step 432, the process 146 resumes, patched entirely live with little to no downtime, within user space. The method terminates at 440.

12

FIG. 5 is a flow diagram illustrating a method 500 patching the process in accordance with another aspect of the disclosure. The patcher 116 is an exemplary implementation of the method 500 as executed on computer system 20 by CPU 21 shown in FIG. 8.

Method 500 begins at 502 and proceeds to 504, where the patcher 116 interprets commands and arguments, for example, from a command line, or other user interface. At 504, if the command is a “list” command, the method proceeds to 510 where applied patches are listed out by the patcher 116 via the user interface. The method terminates at 530.

At 506, if the command was a “check” command, the method proceeds to 512 where the patcher 116 determines if the patch has already been applied. The process terminates at 530.

At 508, if the patcher 116 does not receive a “revert” command, the patcher 116 has received a “patch” command, and the method proceeds to the steps of method 400 show in FIG. 4. Otherwise, the method 500 proceeds to 514. At 514, the patcher 116 determines whether the patch is applied, and if the patch is not applied, the method proceeds to 530 where the method terminates. Alternatively, after step 514, the method may proceed to 520 if the process was suspended prior to step 506, 508, 512 or 514; or suspends the process just before 516.

If the patch is applied at 514, the method proceeds to 516, where the control transfers performed in method 400 are reversed. The patch itself in address range 300 is unloaded (or, unmapped) at 518. Finally, the process 146 is resumed at 520 and the method terminates at 530.

FIG. 6 shows the details of the binary file 115, in accordance with one aspect of the disclosure. Binary file 115 may comprise binary code (from 112) that is compiled from modified source patch 108, and has first instrumentation applied. Binary file 112 may contain new functions that are new version of functions of the unpatched application. Binary file 112 may also declare one or more symbols that are defined in the unpatched application, e.g., in the binary code 148 of process 146, but not defined in the binary file 112. For example, the binary file 112 identifies the symbol “symbol_2” as a binding symbol (e.g., global or/and external), indicating that during patching these may be resolved. Additionally, the binary file 115 may contain the metadata 600 that is added during the second instrumentation step performed by generator 104. For example, the metadata may contain information such as the offset of symbol_2 in the binary code 148.

FIG. 7 is a flow diagram for a method 700 for generating a binary patch in accordance with an aspect of the disclosure.

The method 700 is an exemplary implementation of the generator 104 of system 100 shown in FIG. 1, as executed by a computer system such as system 20 shown in FIG. 8.

The method begins at 702 and proceeds to 704. At 704, the generator 104 may generate modified source code containing changed function by identifying differences between un-patched source code of an application and a patched version of the application. The modified source code comprises at least definitions of changed or new functions and declarations of or references to the symbols that are used in the patch, but are not defined there.

At 706, a source code instrumentation is performed to declare a portion of the symbols in the modified source code as global and/or external.

US 10,795,659 B1

13

At **708**, the generator **104** may invoke a compiler to compile the modified source code into an executable file, e.g. binary file **112**.

At **710**, the generator **104** performs a binary instrumentation by calculating offsets for the symbols in the binary code **148** as described with respect to FIGS. 1-4 and 6.

Finally, at **712**, the generator **104** includes the offsets as metadata to the binary file forming the binary patch. The method terminates at **720**. In some aspects, portions of method **700** may be optional, such as steps **704-708**, in which case the modified source code is already generated and/or a compiled version of the modified source code is received at the generator **104**.

FIG. 8 illustrates a block diagram of a general-purpose computer system on which the disclosed system and method can be implemented according to an exemplary aspect. It should be noted that the computer system **20** can correspond to the computer **140**, or computers that execute the generator **104** and patcher **116**.

As shown, the computer system **20** (which may be a personal computer or a server) includes a central processing unit **21**, a system memory **22**, and a system bus **23** connecting the various system components, including the memory associated with the central processing unit **21**. As will be appreciated by those of ordinary skill in the art, the system bus **23** may comprise a bus memory or bus memory controller, a peripheral bus, and a local bus that is able to interact with any other bus architecture. The system memory may include permanent memory (ROM) **24** and random-access memory (RAM) **25**. The basic input/output system (BIOS) **26** may store the basic procedures for transfer of information between elements of the computer system **20**, such as those at the time of loading the operating system with the use of the ROM **24**.

The computer system **20** may also comprise a hard disk **27** for reading and writing data, a magnetic disk drive **28** for reading and writing on removable magnetic disks **29**, and an optical drive **30** for reading and writing removable optical disks **31**, such as CD-ROM, DVD-ROM and other optical media. The hard disk **27**, the magnetic disk drive **28**, and the optical drive **30** are connected to the system bus **23** across the hard disk interface **32**, the magnetic disk interface **33** and the optical drive interface **34**, respectively. The drives and the corresponding computer information media are power-independent modules for storage of computer instructions, data structures, program modules and other data of the computer system **20**.

An exemplary aspect comprises a system that uses a hard disk **27**, a removable magnetic disk **29** and a removable optical disk **31** connected to the system bus **23** via the controller **55**. It will be understood by those of ordinary skill in the art that any type of media **56** that is able to store data in a form readable by a computer (solid state drives, flash memory cards, digital disks, random-access memory (RAM) and so on) may also be utilized.

The computer system **20** has a file system **36**, in which the operating system **35** may be stored, as well as additional program applications **37**, other program modules **38**, and program data **39**. A user of the computer system **20** may enter commands and information using keyboard **40**, mouse **42**, or any other input device known to those of ordinary skill in the art, such as, but not limited to, a microphone, joystick, game controller, scanner, etc. . . . Such input devices typically plug into the computer system **20** through a serial port **46**, which in turn is connected to the system bus, but those of ordinary skill in the art will appreciate that input devices may be also be connected in other ways, such as,

14

without limitation, via a parallel port, a game port, or a universal serial bus (USB). A monitor **47** or other type of display device may also be connected to the system bus **23** across an interface, such as a video adapter **48**. In addition to the monitor **47**, the personal computer may be equipped with other peripheral output devices (not shown), such as loudspeakers, a printer, etc.

Computer system **20** may operate in a network environment, using a network connection to one or more remote computers **49**. The remote computer (or computers) **49** may be local computer workstations or servers comprising most or all of the aforementioned elements in describing the nature of a computer system **20**. Other devices may also be present in the computer network, such as, but not limited to, routers, network stations, peer devices or other network nodes.

Network connections can form a local-area computer network (LAN) **50** and a wide-area computer network (WAN). Such networks are used in corporate computer networks and internal company networks, and they generally have access to the Internet. In LAN or WAN networks, the personal computer **20** is connected to the local-area network **50** across a network adapter or network interface **51**. When networks are used, the computer system **20** may employ a modem **54** or other modules well known to those of ordinary skill in the art that enable communications with a wide-area computer network such as the Internet. The modem **54**, which may be an internal or external device, may be connected to the system bus **23** by a serial port **46**. It will be appreciated by those of ordinary skill in the art that said network connections are non-limiting examples of numerous well-understood ways of establishing a connection by one computer to another using communication modules.

In various aspects, the systems and methods described herein may be implemented in hardware, software, firmware, or any combination thereof. If implemented in software, the methods may be stored as one or more instructions or code on a non-transitory computer-readable medium. Computer-readable medium includes data storage. By way of example, and not limitation, such computer-readable medium can comprise RAM, ROM, EEPROM, CD-ROM, Flash memory or other types of electric, magnetic, or optical storage medium, or any other medium that can be used to carry or store desired program code in the form of instructions or data structures and that can be accessed by a processor of a general purpose computer.

In various aspects, the systems and methods described in the present disclosure can be addressed in terms of modules. The term "module" as used herein refers to a real-world device, component, or arrangement of components implemented using hardware, such as by an application specific integrated circuit (ASIC) or field-programmable gate array (FPGA), for example, or as a combination of hardware and software, such as by a microprocessor system and a set of instructions to implement the module's functionality, which (while being executed) transform the microprocessor system into a special-purpose device. A module may also be implemented as a combination of the two, with certain functions facilitated by hardware alone, and other functions facilitated by a combination of hardware and software. In certain implementations, at least a portion, and in some cases, all, of a module may be executed on the processor of a general purpose computer (such as the one described in greater detail in FIG. 7, above). Accordingly, each module may be realized in a variety of suitable configurations, and should not be limited to any particular implementation exemplified herein.

US 10,795,659 B1

15

In the interest of clarity, not all of the routine features of the aspects are disclosed herein. It would be appreciated that in the development of any actual implementation of the present disclosure, numerous implementation-specific decisions must be made in order to achieve the developer's specific goals, and these specific goals will vary for different implementations and different developers. It is understood that such a development effort might be complex and time-consuming, but would nevertheless be a routine undertaking of engineering for those of ordinary skill in the art, having the benefit of this disclosure.

Furthermore, it is to be understood that the phraseology or terminology used herein is for the purpose of description and not of restriction, such that the terminology or phraseology of the present specification is to be interpreted by the skilled in the art in light of the teachings and guidance presented herein, in combination with the knowledge of the skilled in the relevant art(s). Moreover, it is not intended for any term in the specification or claims to be ascribed an uncommon or special meaning unless explicitly set forth as such.

The various aspects disclosed herein encompass present and future known equivalents to the known modules referred to herein by way of illustration. Moreover, while aspects and applications have been shown and described, it would be apparent to those skilled in the art having the benefit of this disclosure that many more modifications than mentioned above are possible without departing from the inventive concepts disclosed herein.

What is claimed is:

1. A system for applying a patch to a running process in user space comprising:
 - a process executing in user space in an operating system executed by a hardware processor; and
 - a patcher configured to:
 - suspend execution of the process, wherein a memory address space of the process contains binary code executed in the process, and wherein the binary code comprises one or more symbols;
 - map a binary patch to the memory address space of the process, wherein the binary patch contains amendments to the binary code, wherein the binary patch references a portion of the one or more symbols, and wherein the binary patch contains metadata indicating offsets of the portion of the one or more symbols, wherein the patcher is configured to map the binary patch by:
 - injecting parasite code into the process;
 - transferring control to the parasite code; and
 - mapping, using the parasite code, the binary patch to the memory address space by executing instructions in the parasite code;
 - resolve the portion of the one or more symbols using the offsets in the metadata; and
 - resume execution of the process.
2. The system of claim 1, wherein the patcher is configured to resolve symbols by:
 - finding an address of at least one symbol referenced by the binary patch, in the binary code executed in the process; and
 - linking the at least one symbol by writing the address that was found into a specified location in the binary patch that is mapped into the memory address space of the process.
3. The system of claim 2, wherein the patcher is configured to find the address by:
 - calculating the address of the at least one symbol by adding an offset taken from metadata stored in the

16

binary patch to the address of a beginning of the binary code executed in the process in the memory address space of the process.

4. The system of claim 2, wherein the patcher is further configured to:
 - determine when there is no metadata information in the binary patch associated with the one or more symbols; and
 - import the one or more symbols from a library, wherein the library is mapped to the memory address space of the process, when the library is not yet mapped.
5. The system of claim 1, wherein the patcher is further configured to:
 - amend binary information of the process to allow control transfer from instructions contained in the executing process prior to the patch, to instructions contained in the binary patch.
6. The system of claim 1, wherein the patcher is further configured to:
 - resume execution of the process, without mapping the binary patch and without resolving symbols in the binary patch, when the process is executing a portion of the associated binary code that contains symbols that are amended in the binary patch; and
 - wait until the patch is applied safely.
7. The system of claim 1, wherein the patcher applies the patch when, at the moment when execution of the process was suspended, the process was not executing functions for which the binary patch contains amendments.
8. A method for live patching a process in user space comprising:
 - stopping execution of the process executing in user space on a hardware processor, wherein a memory address space of the process contains binary code executed in the process;
 - mapping a binary patch to the memory address space of the process, wherein the binary patch contains amendments to the executable binary code, wherein mapping comprises:
 - injecting parasite code into the process;
 - transferring control to the parasite code; and
 - mapping the binary patch to the memory address space by executing instructions in the parasite code;
 - resolving symbols used by the binary patch using metadata regarding the symbols, the metadata stored in the binary patch; and
 - resuming execution of the process when the patching is complete, or patching has failed.
9. The method of claim 8, further comprising:
 - resuming execution of the process, without mapping the binary patch and without resolving symbols in the binary patch, when the process is executing a portion of the associated binary code that contains symbols that are amended in the binary patch; and
 - performing the steps of the method again until the patch is applied safely.
10. The method of claim 8, wherein the patch is applied only when, at the moment when execution of the process was suspended, the process was not executing functions for which the binary patch contains amendments.
11. The method of claim 8, wherein stopping execution of the process further comprises using a debugging interface of an operating system (OS) in which the process is executing.
12. The method of claim 8, wherein the binary patch contains binary code of amended functions of the corresponding binary code.

US 10,795,659 B1

17

13. The method of claim 10, wherein the metadata in the patch contain for at least one symbol used in the binary patch, an offset of the at least one symbol from the beginning of the executable binary code.

14. The method of claim 8, wherein resolving symbols further comprises:

finding an address of at least one symbol referenced by the binary patch, in the executable binary code; and

linking the at least one symbol by writing the address that was found into a specified location in the binary patch that is mapped into the address space of the process.

15. The method of claim 8, wherein finding the address comprises:

calculating the address of the at least one symbol by adding an offset taken from metadata stored in the binary patch to the address of a beginning of the executable binary code in the memory address space of the process.

16. The method of claim 8, further comprising:

determining when there is no metadata information in the binary patch associated with the at least one symbol; and

importing the at least one symbol from a library, wherein the library is mapped to the memory address space of the process when it is not yet mapped.

17. The method of claim 8, further comprising:

amending binary information of the process to allow control transfer from instructions contained in the executing process prior to the patch, to instructions contained in the binary patch.

18. The method of claim 14, wherein the binary information is the executable binary code executed by the process.

19. The method of claim 8, wherein resolving the symbols comprises:

finding an address of an external symbol; and

binding the symbol by writing the address into a specified location in the address space of the process executing in user space.

18

20. A non-transitory computer-readable medium storing therein instructions for executing a method for live patching a process in user space, the instructions comprising:

stopping execution of the process executing in user space on a processor, wherein a memory address space of the process contains executable binary code executed by the process;

mapping a binary patch to the memory address space of the process, wherein the binary patch contains amendments to the executable binary code, wherein mapping comprises:

injecting parasite code into the process;

transferring control to the parasite code; and

mapping the binary patch to the memory address space by executing instructions in the parasite code;

resolving symbols used by the binary patch using metadata regarding the symbols, the metadata stored in the binary patch; and

resuming execution of the process when the patching is complete, or patching has failed.

21. The non-transitory computer-readable medium of claim 20, the instructions further comprising instructions for:

resuming execution of the process, without mapping the binary patch and without resolving symbols in the binary patch, when the process is executing a portion of the associated binary code that contains symbols that are amended in the binary patch; and

performing the steps of the method again until the patch is applied safely.

22. The non-transitory computer-readable medium of claim 20, the instructions for resolving symbols further comprises instructions for:

finding an address of at least one symbol referenced by the binary patch, in the executable binary code; and

linking the at least one symbol by writing the address that was found into a specified location in the binary patch that is mapped into the address space of the process.

* * * * *

EXHIBIT E



US008145740B1

(12) **United States Patent**
Tormasov et al.

(10) **Patent No.:** **US 8,145,740 B1**
(45) **Date of Patent:** ***Mar. 27, 2012**

(54) **VIRTUAL COMPUTING ENVIRONMENT**

(75) Inventors: **Alexander G. Tormasov**, Moscow (RU);
Stanislav S. Protasov, Moscow (RU);
Serguei M. Belousov, Singapore (SG);
Dennis Lunev, Moscow (RU)

(73) Assignee: **Parallels Holdings, Ltd.** (BM)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 887 days.

This patent is subject to a terminal disclaimer.

(21) Appl. No.: **12/189,054**

(22) Filed: **Aug. 8, 2008**

Related U.S. Application Data

(63) Continuation of application No. 11/279,902, filed on Apr. 17, 2006, now Pat. No. 7,426,565, which is a continuation of application No. 09/918,031, filed on Jul. 30, 2001, now Pat. No. 7,099,948.

(60) Provisional application No. 60/269,655, filed on Feb. 16, 2001.

(51) **Int. Cl.**
G06F 15/173 (2006.01)

(52) **U.S. CL.** **709/223; 709/228; 709/229; 707/10**

(58) **Field of Classification Search** **709/223, 709/228, 229; 707/10**
See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

6,779,016 B1 * 8/2004 Aziz et al. 709/201
* cited by examiner

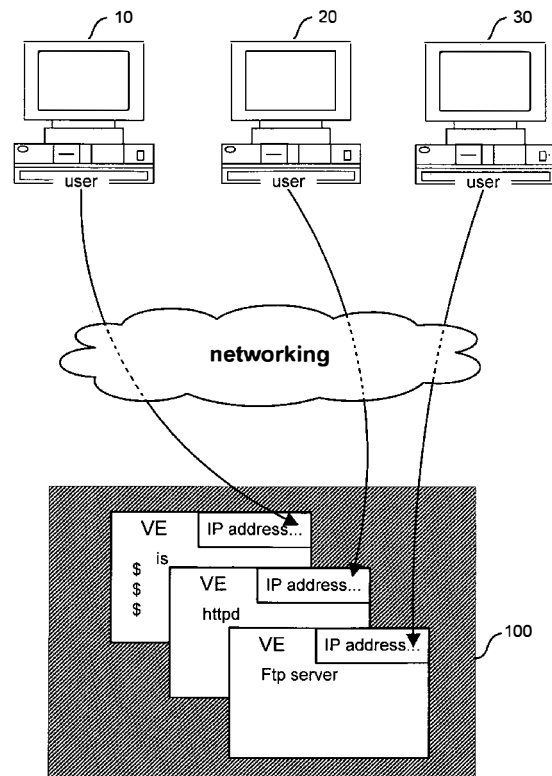
Primary Examiner — Adnan Mirza

(74) *Attorney, Agent, or Firm* — Bardmesser Law Group

(57) **ABSTRACT**

A computing system includes a physical server having a single instance of an operating system; and a plurality of virtual environments running on the physical server and directly supported by the single instance of the operating system. Each virtual environment responds to requests from users and appears to the users as a stand-alone server having its own instance of the operating system. Each virtual environment has a plurality of objects associated with it and supported by the operating system. Some of the objects are private and other objects are shared between multiple virtual environments. One virtual environment cannot access private objects of another virtual environment.

24 Claims, 3 Drawing Sheets



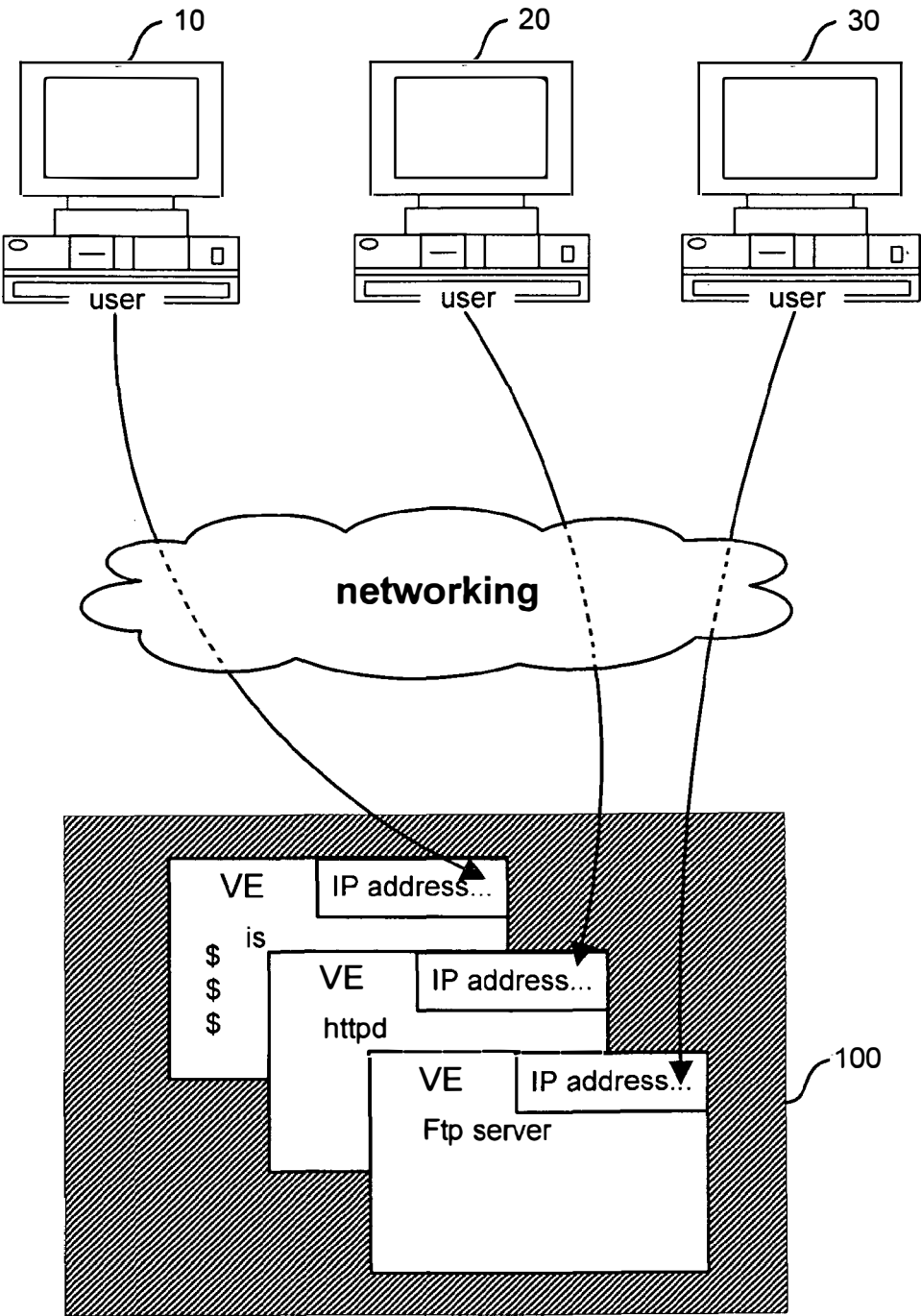


FIG. 1

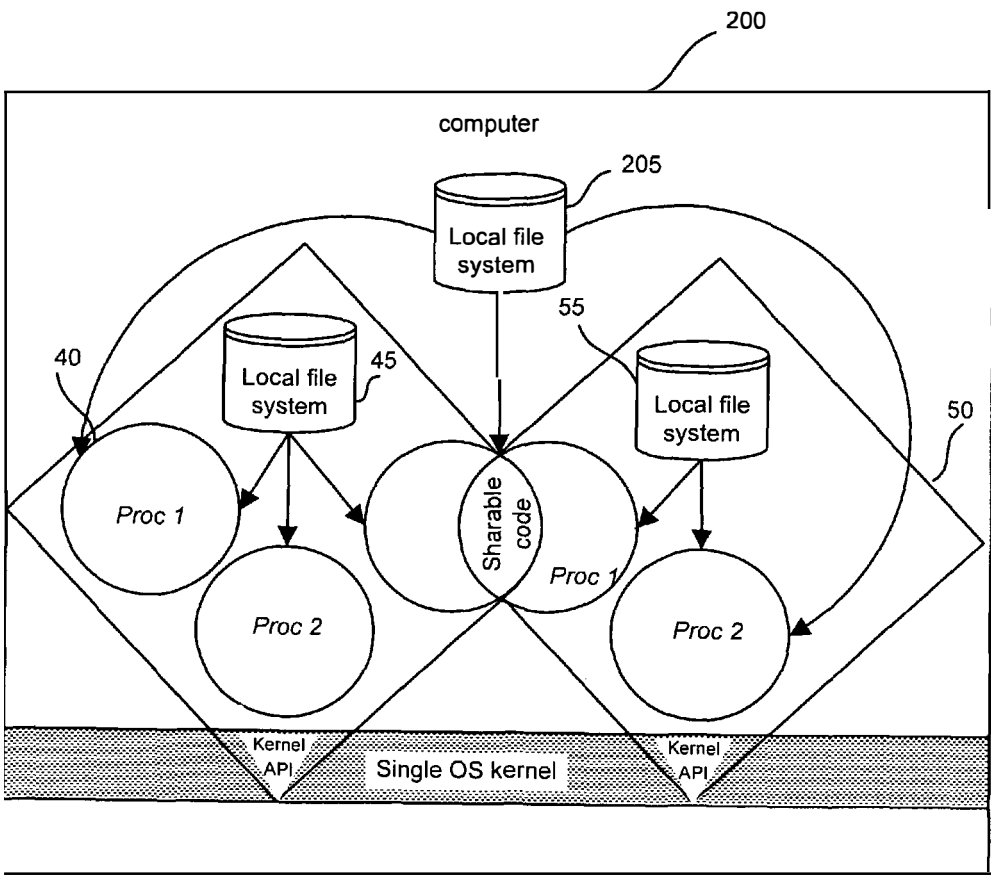


FIG. 2

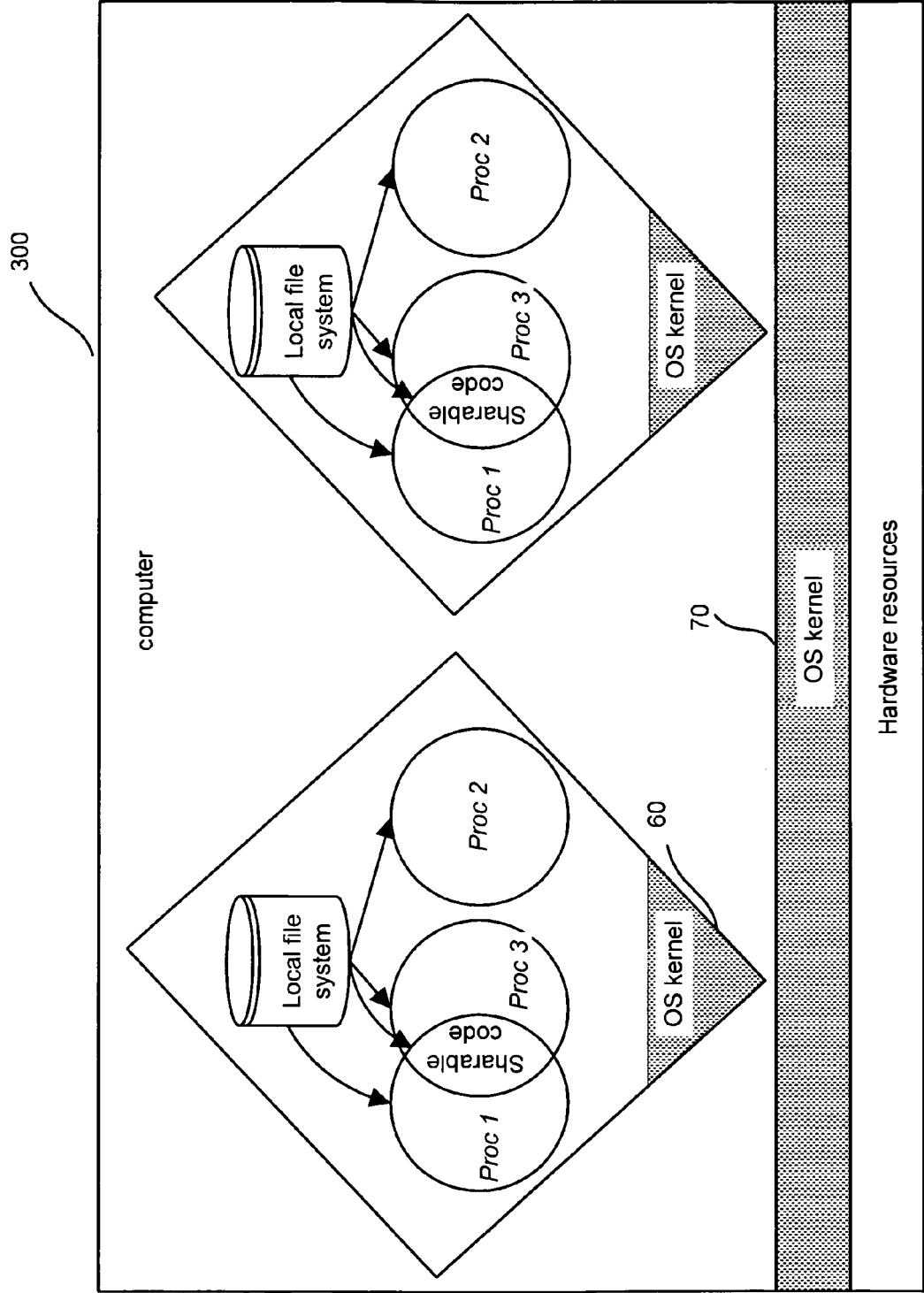


FIG. 3

US 8,145,740 B1

1

VIRTUAL COMPUTING ENVIRONMENT**CROSS-REFERENCE TO RELATED APPLICATIONS**

This application is a continuation of U.S. patent application Ser. No. 11/279,902, filed on Apr. 17, 2006, entitled VIRTUAL COMPUTING ENVIRONMENT, which is a continuation of U.S. patent application Ser. No. 09/918,031, filed on Jul. 30, 2001, entitled VIRTUAL COMPUTING ENVIRONMENT, which claims the benefit of U.S. Provisional Application for Patent No. 60/260,655 entitled USE OF VIRTUAL COMPUTING ENVIRONMENTS TO PROVIDE FULL INDEPENDENT OPERATING SYSTEM SERVICES ON A SINGLE HARDWARE NODE, filed on Feb. 16, 2001, which are both incorporated by reference herein in their entirety.

FIELD OF THE INVENTION

This invention relates to the provision of full independent computer system services across a network of remote computer connections.

DESCRIPTION OF THE PRIOR ART

The problem of providing computer services across remote computer connections has existed during the last 30-40 years, beginning with the early stages of computer technologies. In the very beginning, during the mainframe computer age, this problem was solved by renting computer terminals which were associated with a mainframe computer and then connecting the related computer terminals to the mainframe computer using a modem or dedicated lines to provide the mainframe computer with data access services, see, e.g., U.S. Pat. No. 4,742,477 to Bach, 1987. Later, with the beginning of the age of personal computers and with the widespread acceptance of the client-server model. The problem of access to large information sources in the form of computer readable data, at first glance, seems to have been solved. Specifically, every user could have his own computer and then rent an Internet connection to obtain access to information sources or data stored on other computers.

Today, with wide growth of Internet access, another problem has arisen—the problem of information creation. Usually, users want to put out their own information sources in the form of websites and then provide other computer users with access to these websites. However, it is not possible to install a web server on most home connections to a personal computer, simply because the connection to the network from a home computer is usually not adequate to handle the amount of data transfer required. Accordingly, this need has given birth to an industry called a “hosting service”—a hosting service provides computer users with an ability to utilize installed web services.

When one wants to provide Internet users with information in the form of computer readable data (usually in web server form) that could be of interest to a wide range of Internet users, one must store the information and provide a reliable network connection to access the information when needed.

The problem of providing ordinary personal computer users access to information on large capacity computers occurred virtually from the beginning of personal computer production. During the era of the mainframe computer, when direct user access to computer equipment was difficult, this problem was solved by providing users with remote terminals directly connected to a single mainframe computer. These

2

remote terminals were used to obtain certain services from mainframe computers. The advantage of using multiple remote terminals with a single mainframe computer was that the user had little trouble accessing both the mainframe computer hardware and, to some extent, the software resident on the mainframe computer. This is because mainframe computer administration has always dealt with installing and updating software.

Later, with the introduction of personal computers, each personal computer user could gain access to computing power directly from his workplace or home. With the advent of Internet access, the needs of host users for large amounts of information and robust operating systems were met.

The client-server model of networking computers provides a system for accessing computer readable data in which a personal computer is designated as the client computer and another computer or a set of computers is designated as the server computer. Access to the server computer is carried out in a remote way covering the majority of needs of the common computer users.

But even the client-server model has some very fundamental drawbacks. Specifically, the high price of servicing many client workplace computers, including the creation of a network infrastructure and the installation and upgrading of software and hardware to obtain bandwidth for client computer network access, is a significant drawback. Additionally, the rapid growth of information on the Internet has produced more users, who in turn continue to add more information to the Internet. The required service to client computers should be provided by a sufficiently powerful server computer (usually a web or www server) that has an access channel to the Internet with corresponding power. Usually, personal computers have enough performance capability to interact with most of the web servers, but the typical network access is usually less productive than what is required. Additionally, most home personal computers cannot provide sufficient reliability and security. Apart from Internet services, the same problems occur when ordinary personal computer users utilize very complex software packages. Users spend a lot of time and effort setting up and administering these complex software packages. To solve these web service problems, a remote web host (usually supported by an ISP, i.e., Internet Service Provider) usually hosts the web servers for the personal computer users. Thus, the personal computer user is restricted to use of the standard preinstalled web server of the ISP. As a result, the personal computer user's options are limited.

Problems usually arise with the use of CGI (Common Gateway Interface) scripts and more complex applications requiring a database. Such computer tools cannot be used to access any of the personal computer user's programs on a remote server. The personal computer user is used to the absolute freedom of adjustment of his local machine, and therefore the limitations that are imposed by the administration of a remote node on a data storage network are often unacceptable.

One solution to these problems is the use of computer emulators. The OS/390 operating system for IBM mainframe computers has been in use for many years. The same products with hardware partitioning are produced by another vendor of computers—Sun Microsystems. Each personal computer user is given a fully-functional virtual computer with emulated hardware. This approach is very costly, because the operating system installed in the corresponding virtual computer does not recognize the existence of the neighboring analogous computers and shares practically no resources with

US 8,145,740 B1

3

those computers. Experience has shown that the price associated with virtual computers is very great.

Another analogous solution for non-mainframe computers utilizes software emulators of the VMware type (see VMware Worldstation 2.0 documentation). These software programs exist for different types of operating systems and wholly emulate a typical computer inside one process of a main computer operating system.

The main problem is the limitation on the number of computer emulators that can be used on a typically configured server. This limitation is usually due to the fact that the size of the emulated memory is close to the size of the memory used by the process or in which the computer emulator works. That is, the number of computer emulators that can be simultaneously used on one server ranges from about 2-3 to about 10-15. All of the above solutions can be classified as multi-kernel implementations of virtual computers, i.e., the simultaneous existence on one physical computer of several operating system kernels that are unaware of each other.

Therefore, when it is necessary for many personal computer users to deal with a hosting computer, each personal computer user must be provided with a complete set of services that the personal computer user can expect from the host; i.e., a complete virtual environment which emulates a complete computer with installed operating system. For an effective use of equipment, the number of computers in a virtual environment installed in one host computer should be at least two to three times larger than the numbers mentioned above.

BRIEF SUMMARY OF THE INVENTION

The present invention describes efficient utilization of a single hardware system with a single operating system kernel. The end user of a personal computer connected to a server system provided with a virtual computing environment that is functionally equivalent to a computer with a full-featured operating system. There is no emulation of hardware or dedicated physical memory or any other hardware resources as is the case in a full hardware emulation-type solution.

The system and method of the present invention is implemented by the separation of user processes on the level of kernel objects/resources namespace and on the basis of access restrictions enforced inside the operating system kernel. A namespace is a collection of unique names, where a name is an arbitrary identifier, usually an integer or a character string. Usually the term "name" is applied to such objects as files, directories, devices, computers, etc. Virtual computing environment processes are never visible to other virtual computing environments running on the same computer. A virtual computing environment root file system is also never visible to other virtual computing environments running on the same computer. The root file system of a virtual computing environment allows the root user of every virtual computing environment to perform file modifications and local operating system parameters configuration.

BRIEF DESCRIPTION OF THE DRAWING FIGURES

A better understanding of the present invention may be had by reference to the drawing figures, wherein:

FIG. 1 shows a network of end users with access to virtual computing environments encapsulated in a computer with a full-featured operating system in accordance with the present invention;

4

FIG. 2 shows a utilization of hardware resources (memory and file system) by different virtual computing environments; and

FIG. 3 shows a utilization of resources of hardware (memory and file system) in another full hardware emulation solution.

DETAILED DESCRIPTION OF THE INVENTION

The disclosed invention provides for efficient utilization of a single hardware system with a single operating system kernel. The utilization of the disclosed system and method is perceived by the personal computer user as if he has obtained full network root access to a common computer with a fully-featured operating system installed on it. Specifically, the end user of a personal computer is provided with a virtual computing environment that is functionally equivalent to a computer with a full-featured operating system.

From the point of view of the end user of a personal computer, each virtual computing environment is the actual remote computer, with the network address, at which the end user can perform all actions allowed for the ordinary computer: the work in command shells, compilation and installation of programs, configuration of network services, work with offices and other applications. As shown in FIG. 1, several different users 10, 20, 30 of personal computers can work with the same hardware node 100 without noticing each other, as if they worked on totally separate computers with no associated hardware.

Each virtual computing environment includes a complete set of processes and files of an operating system that can be modified by the end user. In addition, each end user 10, 20, 30 may stop and start the virtual computing environment in the same manner as with a common operating system. However, all of the virtual computing environments share the same kernel of the operating system. All the processes inside the virtual computing environment are common processes of the operating system and all the resources inherent to each virtual computing environment are shared in the same way as typically happens inside an ordinary single kernel operating system.

FIG. 2 shows the method enabling the coexistence of (in this case) two virtual computing environments 40, 50 on one hardware computer 200. Each of the two virtual computing environments 40, 50 has its own unique file system 45, 55, and each virtual environment can also see the common file system 205. All the processes of all virtual computing environments work from inside the same physical memory. If two processes in different virtual computing environments were started for execution from one file (for example from the shared file system) they would be completely isolated from each other, but use the same set of read-only shared physical memory pages.

In this manner, a highly effective implementation of multiple virtual computing embodiments inside one operating system is achieved. There is no emulation of hardware or dedicated physical memory or another hardware resource.

As shown in FIG. 3, the disclosed invention differs from the other solutions that provide a complete emulation of computer hardware to give the user a full scope virtual computer at a higher cost. This happens because a minimum of 2 actual kernels 60, 70 are performed in the computer 300, one inside the other—the kernel of the main operating system and inside the process, the kernel of the emulated operating system.

The implementation of the kernels of the operating system with the properties necessary for this invention carry out the separation of the personal computer users not on the level of

US 8,145,740 B1

5

hardware but on the level of the namespace, and on the basis of access limitations, implemented inside the kernels of the operating system.

Virtual computing environment processes are never visible to other virtual computing environments running on the same computer. The virtual computing environment root file system is independent and is also never visible to other virtual computing environments running on the same computer. The root file system of the virtual computing environment allows a root user of every virtual computing environment to make file modifications and configure their own local parameters of the operating system.

The changes done in the file system in one virtual computing environment do not influence the file systems in the other virtual computing environments.

The disclosed system and method has been disclosed by reference to its preferred embodiment. Those of ordinary skill in the art will understand that additional embodiments of the disclosed system and method are made possible by the foregoing disclosure. Such additional embodiments shall fall within the scope and meaning of the appended claims.

What is claimed is:

1. A computing system comprising:
a physical server having an instance of a main operating system (OS);
a plurality of virtual environments running on the physical server and supported by the main OS, each appearing to the users as a stand-alone server;
each virtual environment having its own guest OS kernel;
each virtual environment permitting its root user to configure parameters of its instance of the OS kernel;
each virtual environment having a plurality of private objects supported by the main OS,
a plurality of public objects shared between multiple virtual environments and supported by the main OS,
wherein one virtual environment cannot access private objects of another virtual environment, and
wherein two processes from different virtual environments started for execution from one main OS file are isolated from each other but share at least some physical memory pages using hardware capabilities of the processor.
2. The system of claim 1, wherein each virtual environment has an independent root file system.
3. The system of claim 1, wherein each virtual environment has a file system that is invisible to other virtual environment.
4. The system of claim 1, wherein each virtual environment has a complete set of operating system processes.
5. The system of claim 1, wherein each virtual environment has a complete set of operating system files.
6. The system of claim 1, wherein each virtual environment can modify any file of its own instance of the operating system.
7. The system of claim 1, wherein none of the virtual environments have dedicated memory allocated to them.
8. The system of claim 1, wherein none of the virtual environments utilize emulated hardware.
9. The system of claim 1, wherein each object has a corresponding identifier,
wherein at least some of the identifiers are the same for objects associated with different virtual environments, and
wherein objects of different virtual environments are isolated from each other even when they have the same identifiers.

6

10. The system of claim 1, wherein resources of the operating system kernel belonging to different virtual environments are separated on a namespace level.

11. The system of claim 1, wherein resources and objects of one virtual environment are not visible to processes and objects of other virtual environments.

12. The system of claim 1, wherein the virtual environment comprises processes and files of the operating system kernel.

13. A method of operating a computing system comprising:
a physical server having an instance of a main operating system (OS);

a plurality of virtual environments running on the physical server and supported by the main OS, each appearing to the users as a stand-alone server;

each virtual environment having its own guest OS kernel;
each virtual environment permitting its root user to configure parameters of its instance of the OS kernel;

each virtual environment having a plurality of private objects supported by the main OS,

a plurality of public objects shared between multiple virtual environments and supported by the main OS,
wherein one virtual environment cannot access private objects of another virtual environment, and

wherein two processes from different virtual environments started for execution from one main OS file are isolated from each other but share at least some physical memory pages using hardware capabilities of the processor.

14. The method of claim 13, wherein each virtual environment has an independent root file system.

15. The method of claim 13, wherein each virtual environment has a file system that is invisible to other virtual environment.

16. The method of claim 13, wherein none of the virtual environments have dedicated memory allocated to them.

17. The method of claim 13, wherein none of the virtual environments utilize emulated hardware.

18. The method of claim 13, wherein each object has a corresponding identifier,

wherein at least some of the identifiers are the same for objects associated with different virtual environments, and

wherein objects of different virtual environments are isolated from each other even when they have the same identifiers.

19. The method of claim 13, wherein resources of the operating system kernel belonging to different virtual environments are separated on a namespace level.

20. The method of claim 13, wherein resources and objects of one virtual environment are not visible to processes and objects of other virtual environments.

21. The method of claim 13, wherein each virtual environment includes processes and files of the operating system kernel.

22. The method of claim 13, wherein each virtual environment has a complete set of operating system processes.

23. The method of claim 13, wherein each virtual environment has a complete set of operating system files.

24. The method of claim 13, wherein each virtual environment can modify any file of its own instance of the operating system.

* * * * *

EXHIBIT F



US008539137B1

(12) **United States Patent**
Protassov et al.

(10) **Patent No.:** **US 8,539,137 B1**
(45) **Date of Patent:** **Sep. 17, 2013**

(54) **SYSTEM AND METHOD FOR
MANAGEMENT OF VIRTUAL EXECUTION
ENVIRONMENT DISK STORAGE**

2006/0069828 A1* 3/2006 Goldsmith 710/100
2007/0016904 A1* 1/2007 Adlung et al. 718/1
2007/0234342 A1* 10/2007 Flynn et al. 717/174

OTHER PUBLICATIONS

(75) Inventors: **Stanislav S. Protassov**, Moscow (RU);
Alexander G. Tormasov, Moscow (RU);
Serguei M. Belousov, Singapore (SG)

Muller, Al, and Seburn Wilson. Virtualization with VMware™ ESX Server™. 1. Rockland, MA: Syngpress Publishing, Inc., 2005. Print.*

(73) Assignee: **Parallels IP Holdings GmbH**,
Schaffhausen (CH)

Warren, Steven. The VMware workstation 5 handbook. Hingham, MA: Charles River Media, Jun 1, 2005. 352. Print.*

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 1495 days.

JeongWon Kim; SeungWon Lee; KiDong Chung; "Implementation of zero-copy with caching for efficient networked multimedia service in Linux kernel," Multimedia and Expo, 2001. ICME 2001. IEEE International Conference on , vol., no., pp. 1215-1218, Aug. 22-25, 2001.*
What is Xen?, <http://wiki.xensource.com/xenwiki/XenOverview>.*

(21) Appl. No.: **11/757,598**

* cited by examiner

(22) Filed: **Jun. 4, 2007**

Primary Examiner — Sheng-Jen Tsai

Assistant Examiner — Ramon A Mercado

(74) *Attorney, Agent, or Firm* — Bardmesser Law Group

Related U.S. Application Data

(60) Provisional application No. 60/804,384, filed on Jun. 9, 2006.

(51) **Int. Cl.**
G06F 12/00 (2006.01)

(52) **U.S. Cl.**
USPC 711/6; 711/203; 718/104; 719/324

(58) **Field of Classification Search**
USPC 711/6, 203
See application file for complete search history.

ABSTRACT

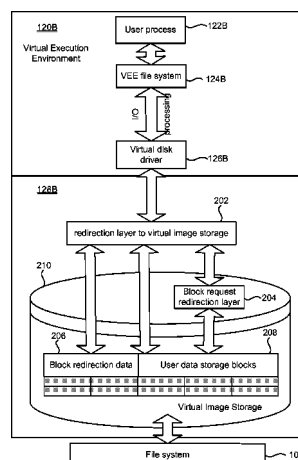
A method, system and computer program product for storing data of a Virtual Execution Environment (VEE), such as a Virtual Private Server (VPS) or a Virtual Machine, including starting an operating system running a computing system; starting a Virtual Machine Monitor under control of the operating system, wherein the VMM virtualizes the computing system and has privileges as high as the operating system; creating isolated Virtual Machines (VMs), running on the computing system simultaneously, wherein each VM executes its own OS kernel and each VM runs under the control of the VMM; starting a storage device driver and a file system driver in the operating system; mounting a virtual disk drive; starting VM-specific file system drivers in the VM, the VM specific file system driver together with the common storage device drivers support virtual disk drives, the virtual disk drive is represented on the storage device as a disk image, the disk image data are stored on the storage device as at least one file that includes user data storage blocks and redirection blocks, the redirection blocks point to user data storage blocks, the redirection blocks have a multilevel hierarchy, and the internal structure is used by the VM-specific file system driver.

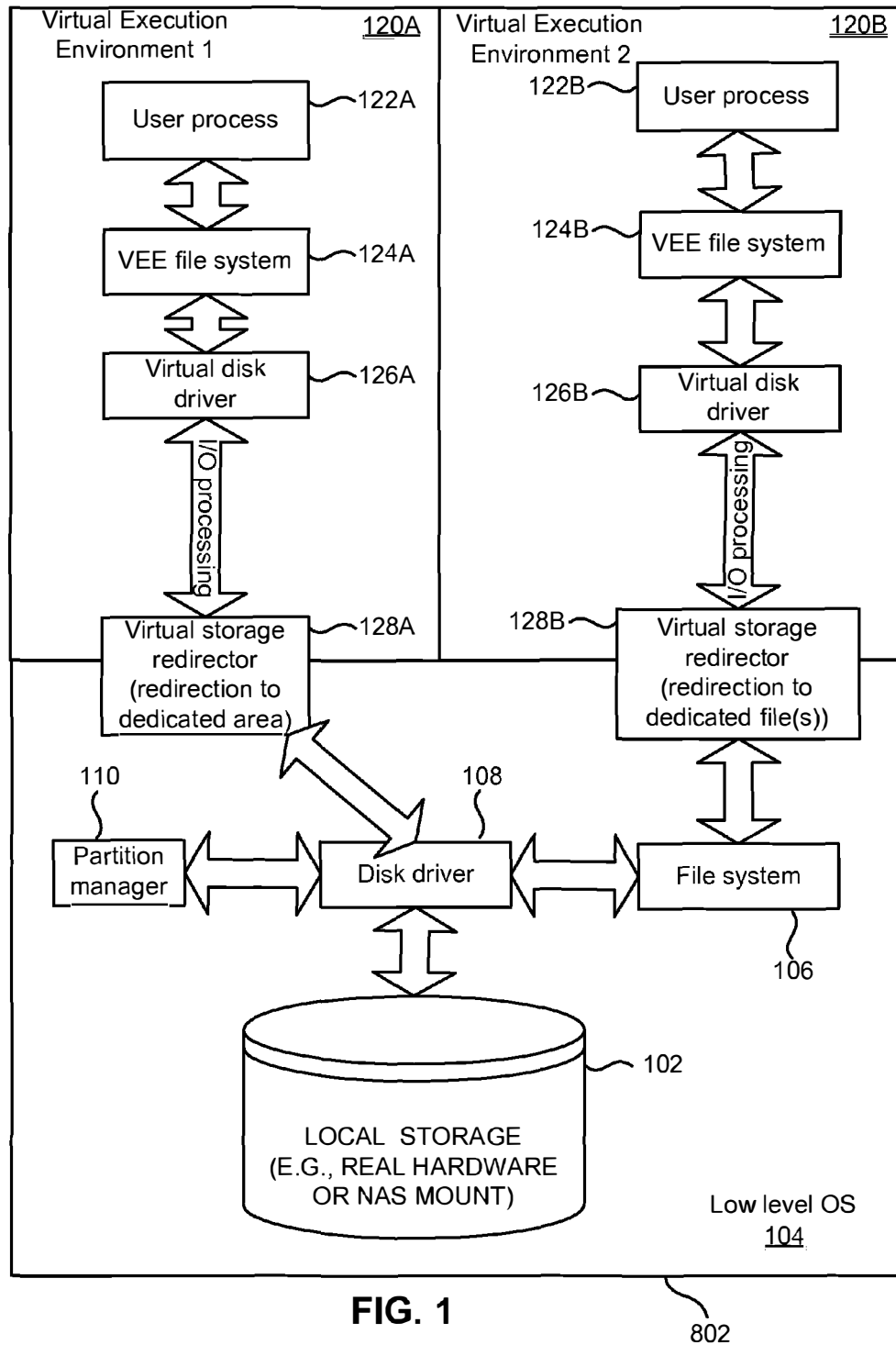
(56) **References Cited**

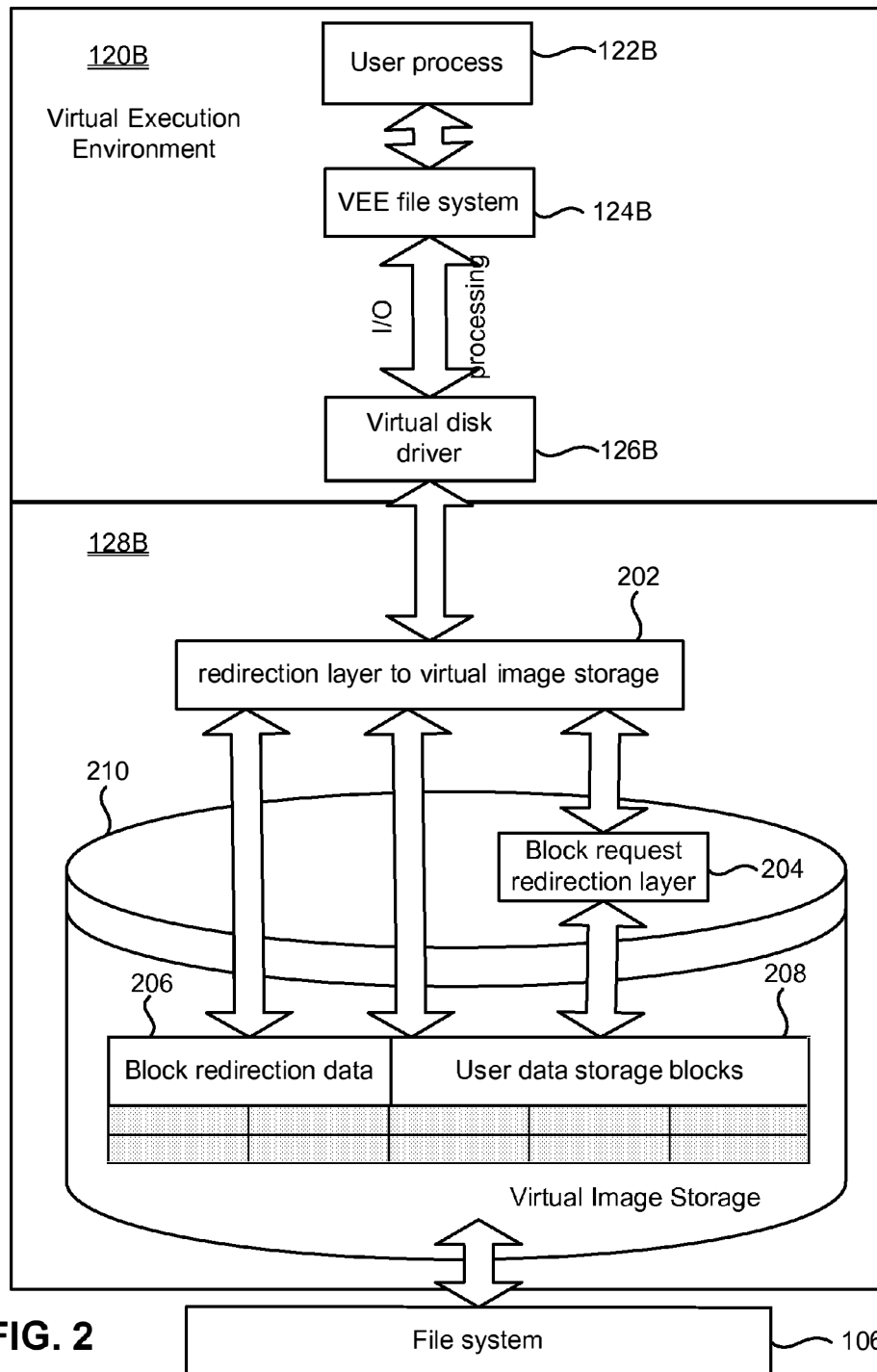
U.S. PATENT DOCUMENTS

5,566,331 A * 10/1996 Irwin et al. 1/1
6,802,062 B1 * 10/2004 Yamada et al. 718/1
6,823,458 B1 * 11/2004 Lee et al. 726/16
6,857,059 B2 * 2/2005 Karpoff et al. 711/209
7,412,702 B1 * 8/2008 Nelson et al. 718/1
7,849,098 B1 * 12/2010 Scales et al. 707/781
7,865,893 B1 * 1/2011 Melyanchuk et al. 718/1
2005/0132365 A1 * 6/2005 Madukkarumukumana et al. 718/1
2005/0268298 A1 * 12/2005 Hunt et al. 718/1
2006/0005189 A1 * 1/2006 Vega et al. 718/1

21 Claims, 8 Drawing Sheets







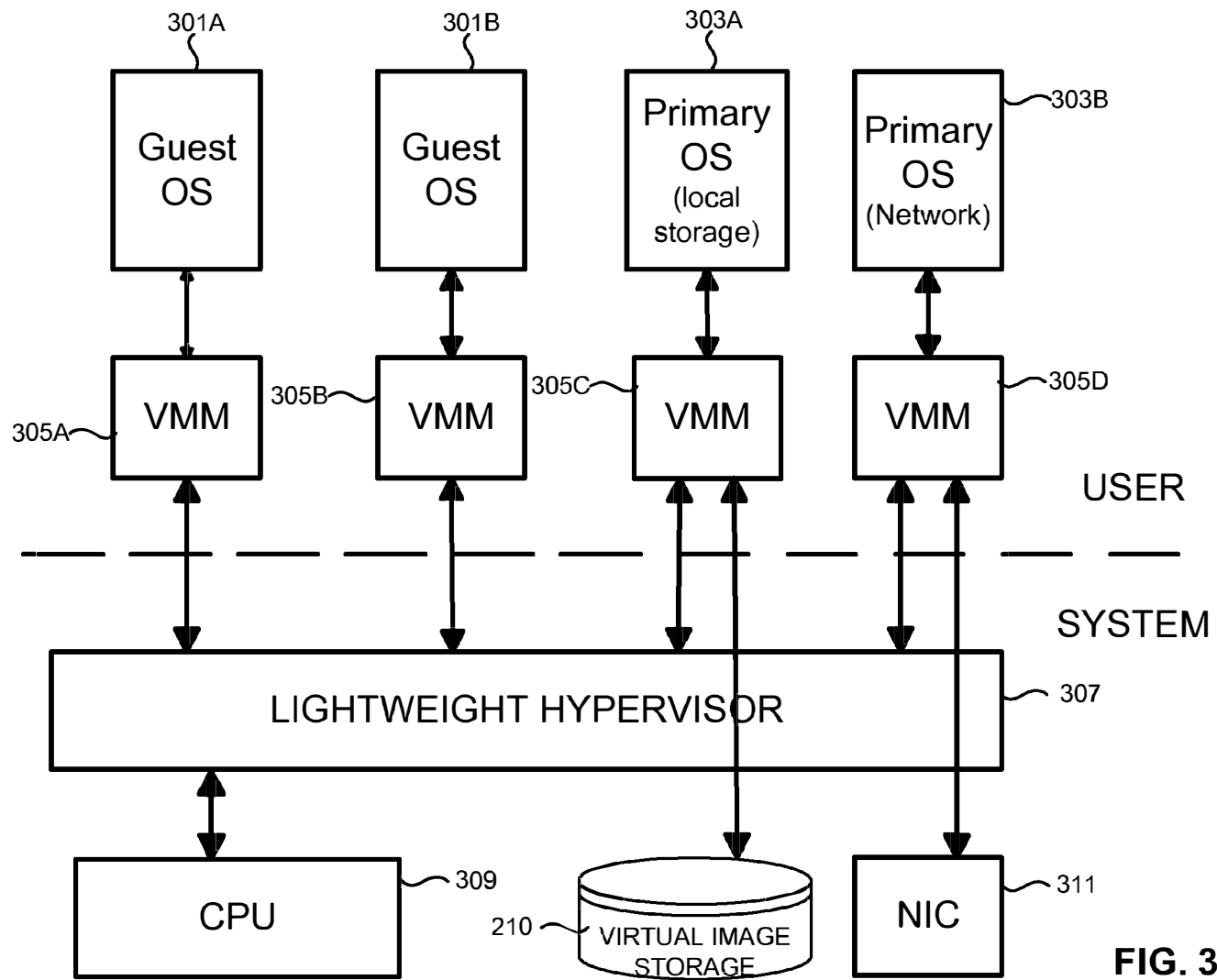


FIG. 3

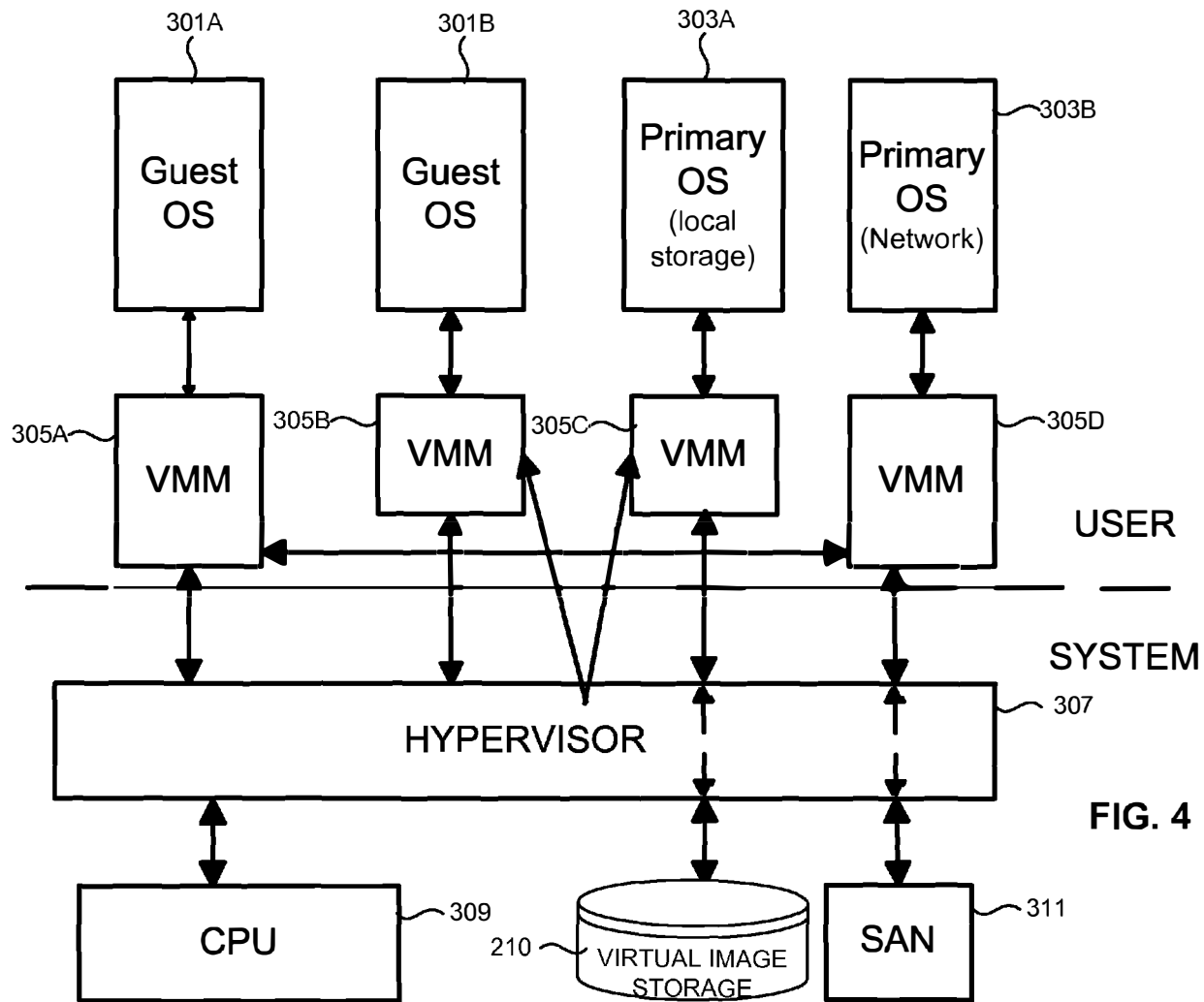


FIG. 4

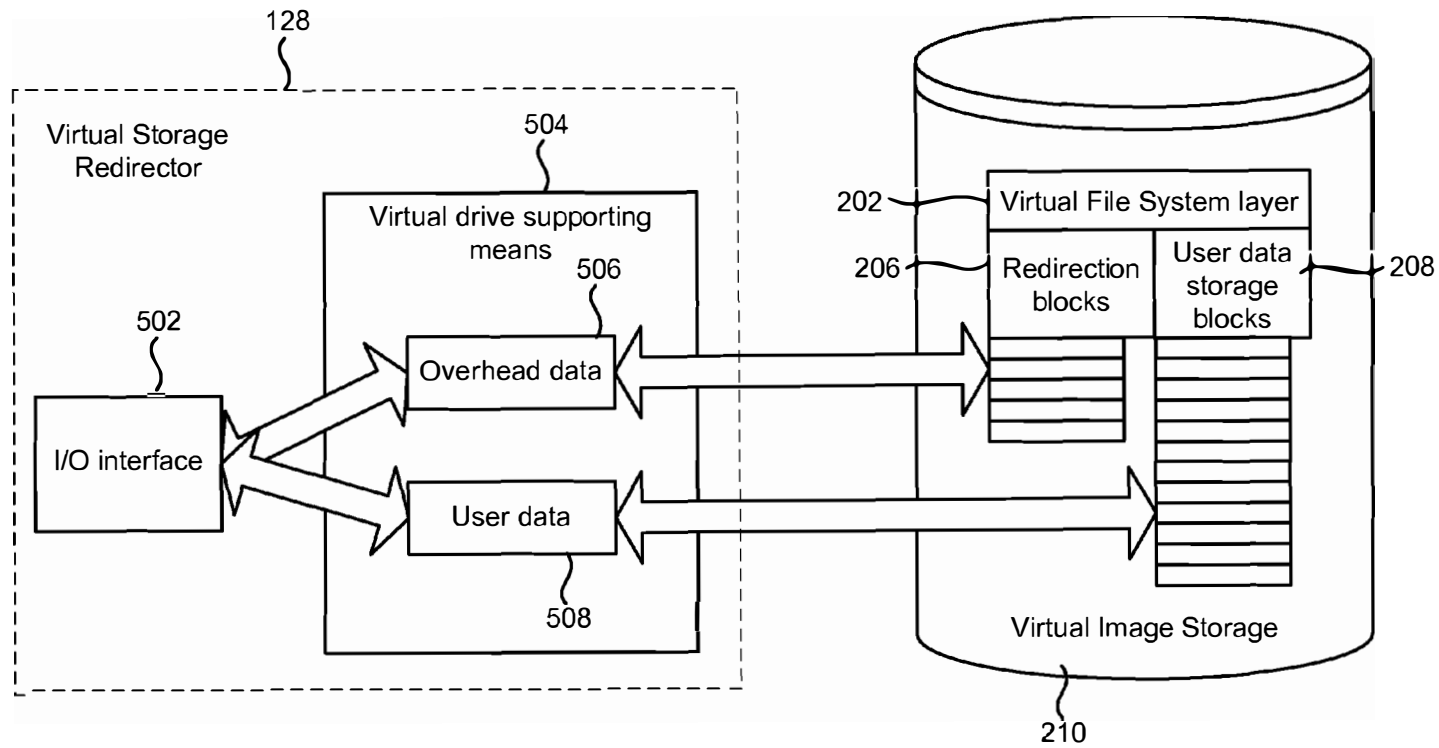


FIG. 5

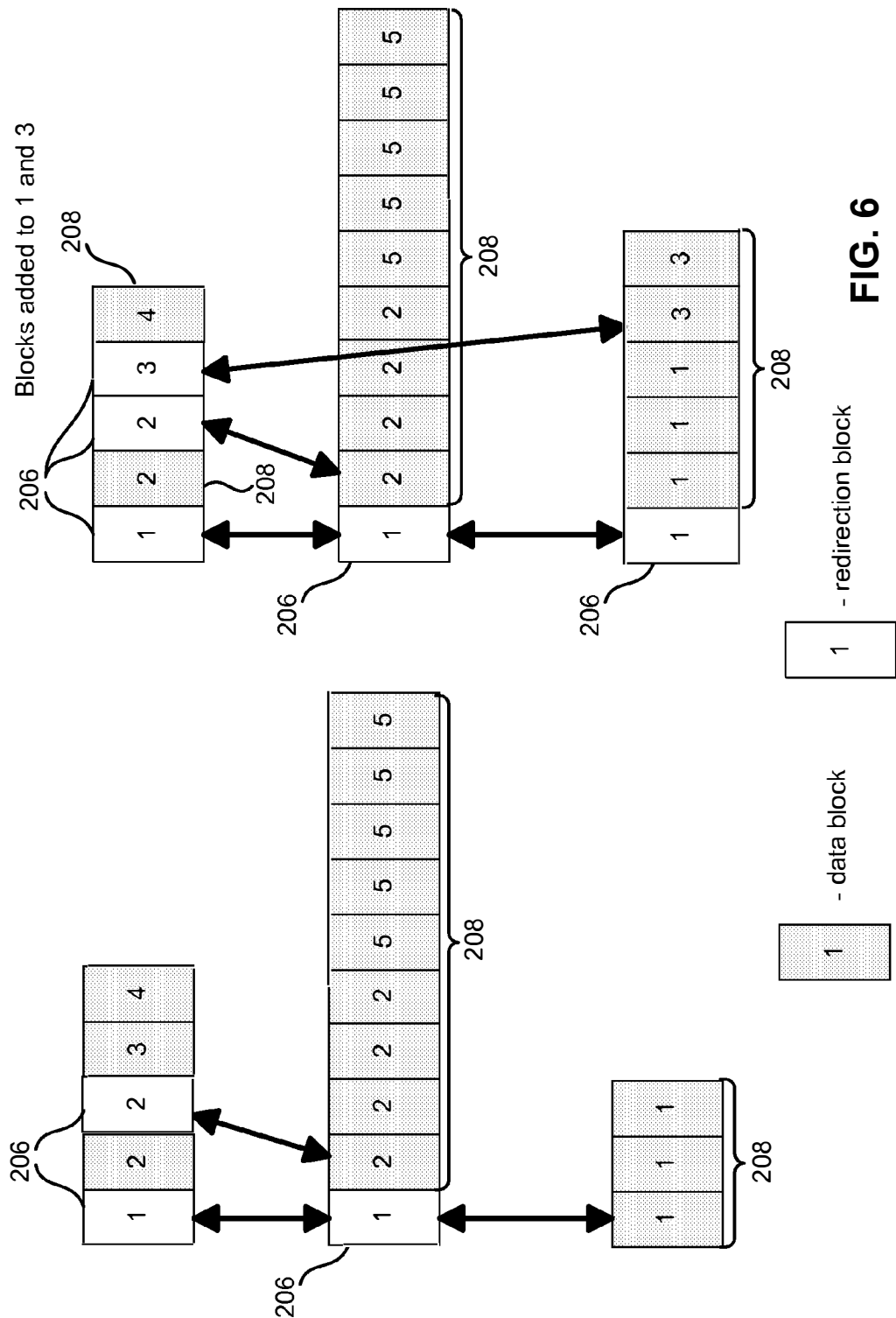


FIG. 6

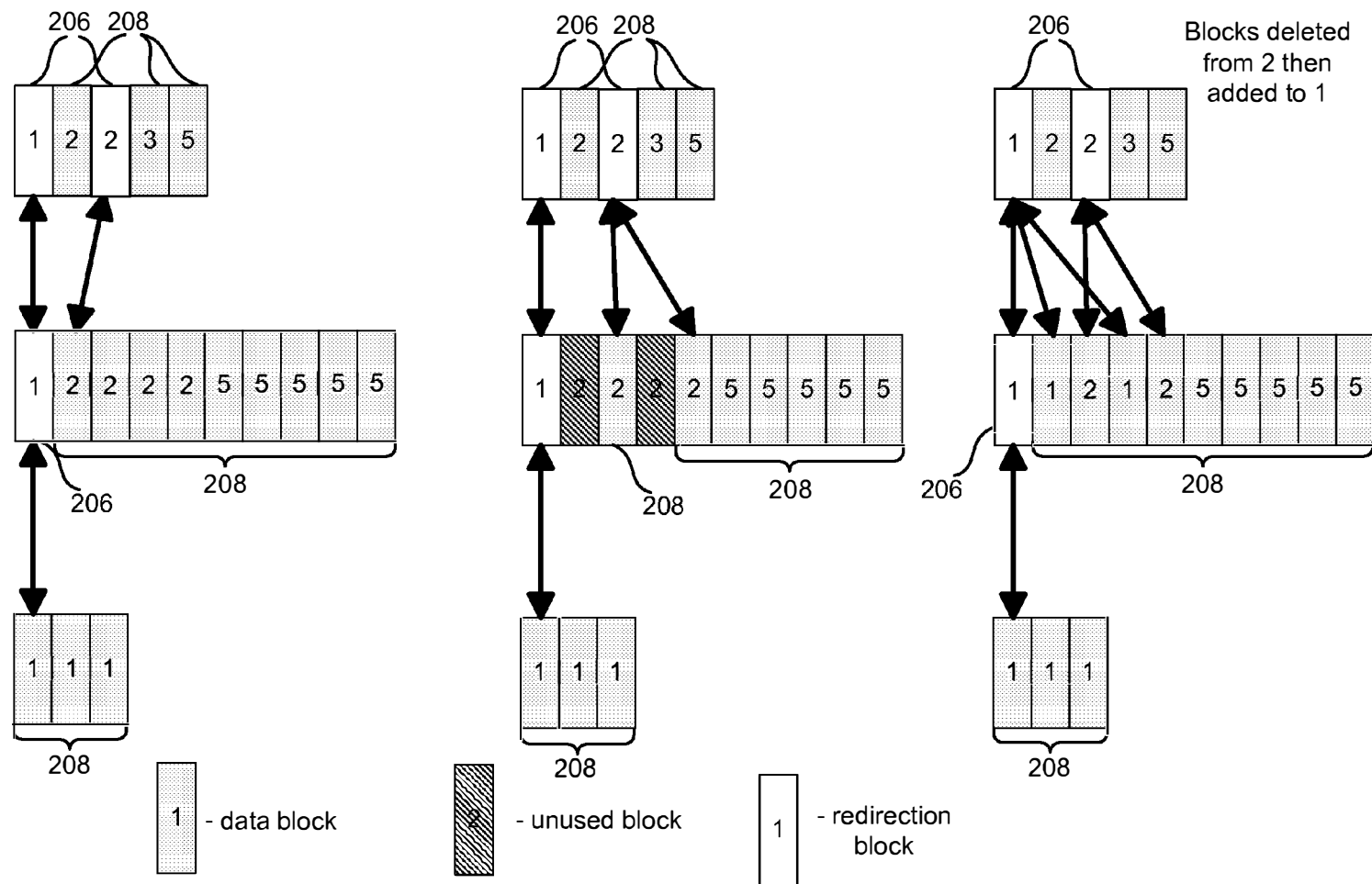


FIG. 7

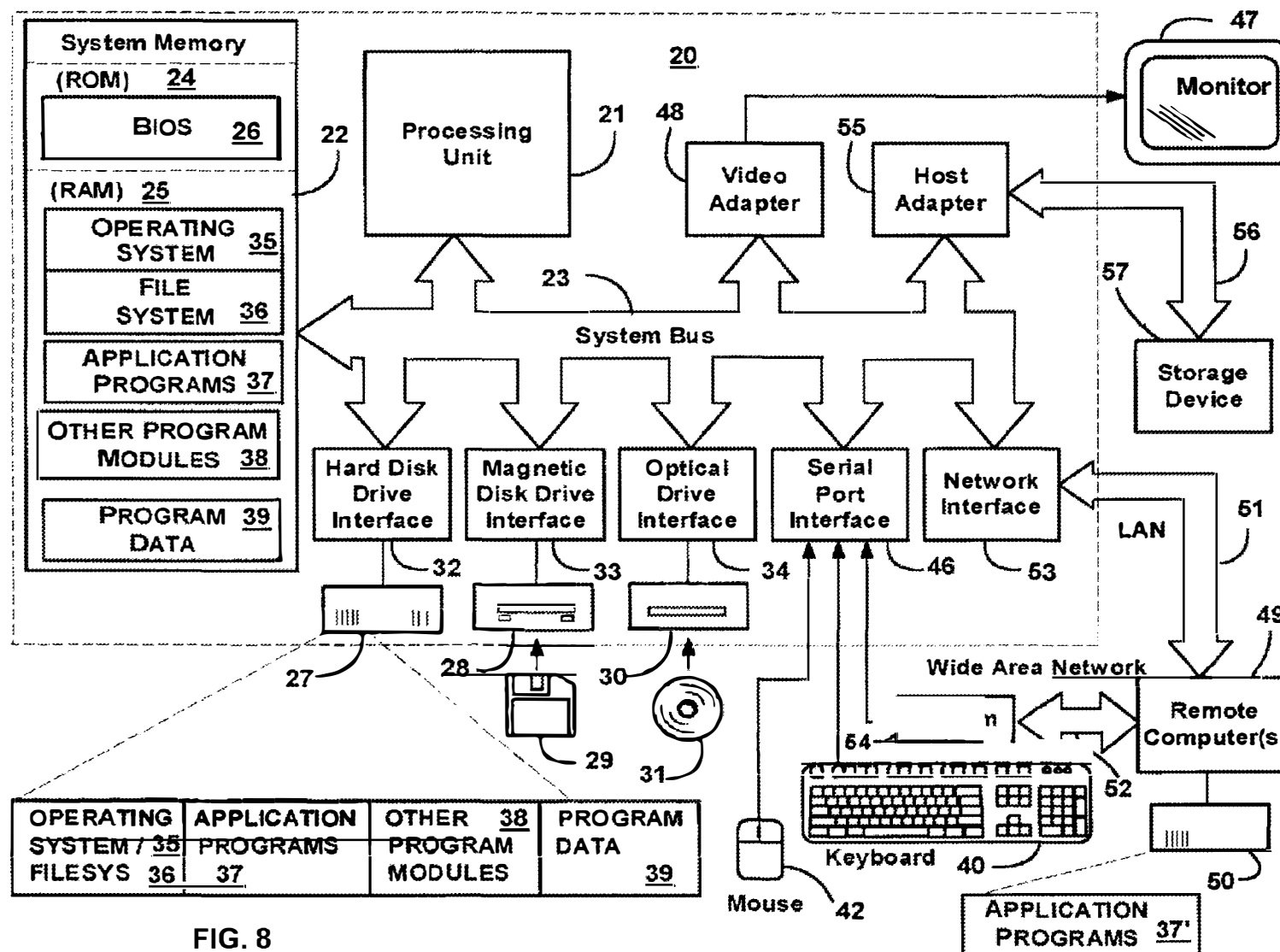


FIG. 8

US 8,539,137 B1

1

SYSTEM AND METHOD FOR MANAGEMENT OF VIRTUAL EXECUTION ENVIRONMENT DISK STORAGE

CROSS-REFERENCE TO RELATED APPLICATIONS

This patent application is a Non-Provisional of U.S. Provisional Patent Application No. 60/804,384; Filed: Jun. 9, 2006, entitled: SYSTEM AND METHOD FOR MANAGEMENT OF VIRTUAL EXECUTION ENVIRONMENT DISK STORAGE, which is incorporated by reference herein in its entirety.

BACKGROUND OF THE INVENTION

1. Field of the Invention

This invention relates to a method, system and computer program product for dynamically managing storage requirements of virtualized systems.

2. Description of the Related Art

Current trends in system and application development today involve ever increasing use of virtualized hardware. For example, servers are often virtualized, as are processors and operating systems. An example of a virtualized server is a Virtual Private Server (VPS or Virtual Environment, VE), such as those supplied by SWsoft, Inc. Other examples of virtualized hardware involve Virtual Machines that virtualize a processor and/or an operating system running on the processor. Examples of such Virtual Machines are marketed by, for example, Parallels Software International, Inc., Microsoft Corporation, and VMware, Inc.

One of the issues that needs to be addressed in such systems is the scalability of storage. Although the price of storage continues to fall, user requirements increase at least at the same rate. For example, a single virtual server or Virtual Machine may be allocated on the order of 10 Gigabytes of storage. 10 Gigabytes, by current (2006) standards, is a relatively manageable amount, since commonly available hard drives that store 100-200 Gigabytes can be purchased for a few hundred dollars, or even less.

However, scalability of this brute force approach presents a problem. While one virtual server or Virtual Machine may require only 10 Gigabytes of storage, 100 such Virtual Machines or virtual servers would require one terabyte of storage. This, while commercially available, is relatively expensive—costing on the order of \$1,000.

There is therefore a need for storage device virtualization for Virtual Execution Environments (VEEs) of different kinds. Since Virtual Execution Environments use generic operating systems, standard procedures for accessing storage devices and presentation of data being stored are also required. At the same time, using storage areas that are dedicated to each environment requires redundant resources and complex technical solutions. Concurrent access to storage device from different Virtual Execution Environments that are invisible to each other may corrupt content of the storage device's service structures, as well as its contents.

The simplest way to organize storage areas for Virtual Execution Environment is to assign a drive partition for each one. This consumes excess storage capacity and does not provide for sharing of data between different VEEs.

Using a set of files for storing data related to virtual storage is preferable, since shared data may be stored as a single file shared among several VEEs.

There are some conventional techniques for handling changed file content:

2

(1) In traditional file systems, a formatted disk or disk partition of a fixed size is used for storing file data in an organized form. Such structures provide a possibility of fast reads and writes of the data, but require a lot of disk space reserved for the file system, since predefined disk storage space should be separated, e.g. in the form of disk partition. For better stability of operating system even after unexpected crashes, journalled file systems are used. The journalled file maintains a log, or journal, of activity that has taken place in the primary data areas of the disk. If a crash occurs, any lost data can be restored, because updates to the metadata in directories and bitmaps have been written to a serial log.

Examples of journalled file systems include, e.g., NTFS, Linux, ext3, reiserfs, IBM Journaled File System 2 (JFS2), open source JFS project for Linux, and Xfs.

(2) VMware full image files, can be managed as they grow.

The image files of this kind at first are smaller than the imaged disk (virtual storage) of a virtual computer (or of the Virtual Machine, VM), and grow when data is added to the virtual storage. Such image files contain a lot of unnecessary data, since blocks of a predefined size could only be added to the image file, and the contents of the block may contain useless data. Also, deletion of the data from the disk image does not reduce the image file. Handling of such images, with their linear structure, is not a trivial task, since structures like B-trees are not supported, and each write to the image file requires converting block address of virtual disk to block address of real disk where image file is stored.

(3) Database transactions, which are used to control data commitment to databases. For example, in standard account procedures, it is necessary to modify several databases simultaneously. Since computers fail occasionally (due to power outages, network outages, and so on) there is a potential for one record to be updated or added, but not the others. To avoid these situations, transactions are used to preserve consistency of the file as one of the goals.

In some databases, for example, MySQL open source database, the main database engine is implemented using B-trees/B+ trees structures. It is used for fast determination of record data placement in files that contain database data.

(4) Encryption software that stores a partition image in file. For example, StrongDisk, and PGPdisk provide this option.

(5) Other software provides the ability to store disk data in archived or modified manner. For example, Acronis True Image™ has the ability to store disk data on a block level.

(6) Windows sparse files with one or more regions of unallocated data in them. In most Unix systems, all files can be treated as sparse files.

Accordingly, there is a need in the art for an effective way to allocate storage in systems running multiple virtualized servers or processors.

SUMMARY OF THE INVENTION

Accordingly, the present invention is related to a system, method and computer program product for managing Virtual Execution Environment storage that substantially obviates one or more of the disadvantages of the related art.

In one aspect, there is provided a system, method and computer program product for storing data of a Virtual Execution Environment (VEE), including launching, on a computer system, isolated VEEs with a shared OS kernel; starting a common storage device driver and a common file system driver; mounting a virtual disk drive for each VEE, wherein each VEE has its own file system drivers for handling a disk image of its VEE; wherein the virtual disk drive is represented on the storage device as a disk image, and wherein the disk

US 8,539,137 B1

3

image is stored on a physical storage device or on a network storage device as at least one file.

In a further aspect, the file can be a generic file whose size is dynamically changed when content of the virtual disk drive is changed. The file of the virtual drive further has an internal structure includes user data storage blocks and redirection blocks that point to user data storage blocks and that are used by the VEE-specific file system driver. At least one file has a B-tree or B+tree structure. The redirection blocks have a multilevel hierarchy. The virtual drive supporting means is running in operating system space and is used by different VEEs. The disk image includes files common to different VEEs. The disk image files are dynamically updated during writes to the virtual disk drive. The disk image files are reduced in size to exclude data corresponding to unused blocks of the virtual disk drive. The disk image files contain only used blocks data. Some disk image files can be shared between VEEs. The drive image files support starting the VEE contents on another hardware system or other VEE. The internal structure of the at least one file provides a possibility of writes to the disk image in a fault tolerant manner; and any virtual disk writes are added to the at least one file with preserving the internal structure integrity in any moment of time. Online or offline snapshotting of the virtual disk drive can be performed using an internal structure of the at least one file.

In another aspect, there is provided a method, system and computer program product for storing data of a Virtual Machine, including starting an operating system running a computing system; starting a Virtual Machine Monitor (VMM) under control of the operating system, wherein the VMM virtualizes the computing system; creating isolated Virtual Machines (VMs), running on the computing system simultaneously, wherein each VM executes its own OS kernel and each VM runs under the control of the VMM; starting a common storage device driver and a common file system driver in the operating system; mounting a virtual disk drive; starting VM-specific file system drivers in the VM. The VM specific file system driver together with the common storage device drivers support virtual disk drives. The virtual disk drive is represented on the storage device as a disk image. The disk image data are stored on the storage device as at least one file that includes user data storage blocks and redirection blocks, the redirection blocks point to user data storage blocks. The redirection blocks have a multilevel hierarchy. The internal structure is used by the VM-specific file system driver. The disk image files can be dynamically updated during writes to the virtual disk drive, and/or can be reduced in size to exclude data corresponding to unused blocks of the virtual disk drive, and/or can contain only used blocks data, and/or can be shared between VEEs.

In another aspect, there is provided a method, system and computer program product for storing data of a Virtual Machine, including starting a Virtual Machine Monitor (VMM) running with system level privileges; creating a primary Virtual Machine (VM) without system level privileges and having a host operating system (HOS) within it, the HOS having direct access to at least some I/O devices; creating a secondary Virtual Machine running without system level privileges; starting a common storage device driver and a common file system driver in the HOS; starting a VM-specific file system driver in the secondary VM; starting a VM-specific virtual device driver in the secondary VM that provides access to the virtual storage drive. The VM specific virtual device driver has access to the disk image files from the virtual device driver via the common file system driver and common storage device driver; and mounting a virtual disk

4

drive. The virtual disk drive is represented on a real storage device as a disk image. The disk image data is stored on the real storage device as at least one file. The file of the virtual drive further has an internal structure includes user data storage blocks and redirection blocks that point to user data storage blocks and that are used by the VEE-specific file system driver.

In another aspect, there is provided a method, system and computer program product for storing data of Virtual Execution Environments, including starting a Virtual Machine Monitor (VMM) with system level privileges; creating a primary Virtual Machine (VM) without system level privileges and having a host operating system (HOS) running within it; starting, a common safe interface providing secure access from the VMs to computing system hardware; creating a secondary Virtual Machine running without system level privilege, wherein the secondary VM uses the common safe interface for accessing hardware; starting a common storage device driver and a common file system driver in the HOS; starting a VM-specific file system driver in the secondary VM; starting a VM-specific virtual device driver in the secondary VM that provides access to the virtual storage drive and has access to the disk image files from the virtual device driver via common file system driver, common storage device driver and the common safe interface; and mounting a virtual disk drive for the secondary VM; The virtual disk drive is represented on a real storage device as a disk image. The disk image data is stored on the real storage device as at least one file.

In another aspect, there is provided a method, system and computer program product for storing data of Virtual Execution Environments, including starting an operating system on a computing system that uses a file system cache area for storing disk data being read or written; starting a Virtual Machine Monitor (VMM) under the control of the operating system; starting isolated Virtual Machines (VMs) on the computing system, wherein each VM executes its own OS kernel and each VM runs under the control of the VMM; starting a common storage device driver and a common file system driver in the operating system; mounting a virtual disk drive, the virtual disk drive provides correct functioning of the VM; starting a VM-specific file system drivers in the VM that, jointly with the common drivers, supports a virtual disk drive that is represented on the storage device as a disk image. The disk image data is stored on the storage device as at least one file. Writes to the file are executed by bypassing the cache area. The internal structure of file enables writes to the disk image in a fault tolerant manner. Any virtual disk writes are added to the file while preserving the internal structure integrity of the file at any moment of time. As will be appreciated, the various aspects described above are applicable to different types of VEEs.

Additional features and advantages of the invention will be set forth in the description that follows, and in part will be apparent from the description, or may be learned by practice of the invention. The advantages of the invention will be realized and attained by the structure particularly pointed out in the written description and claims hereof as well as the appended drawings.

It is to be understood that both the foregoing general description and the following detailed description are exemplary and explanatory and are intended to provide further explanation of the invention as claimed.

BRIEF DESCRIPTION OF THE ATTACHED FIGURES

The accompanying drawings, which are included to provide a further understanding of the invention and are incor-

US 8,539,137 B1

5

porated in and constitute a part of this specification, illustrate embodiments of the invention and together with the description serve to explain the principles of the invention.

In the drawings:

FIG. 1 illustrates an embodiment of the invention where virtual drives are handled using virtual storage redirectors between the Virtual Execution Environments and a low-level operating system area with direct access to local storage.

FIG. 2 illustrates an embodiment of the invention where a virtual storage redirector is responsible for consistency of disk image files and uses a common file system for handling virtual disk reads and writes.

FIG. 3 illustrates an embodiment of a Lightweight Hypervisor Virtual Machine system, where Guest Operating Systems use Primary Operating System's drivers for access to storage drives.

FIG. 4 illustrates an embodiment where Guest Operating Systems use Primary Operating Systems drivers for access to the disk drives when handling virtual storage.

FIG. 5 illustrates an embodiment where the virtual drive supporting means forms file structures corresponding to the virtual drives.

FIG. 6 illustrates an internal structure of the file and time diagram of changing the structure the file when adding data to the file.

FIG. 7 shows changing of the file structure of FIG. 6 during deletion of blocks of the virtual drive in the file and using those blocks for writing new data.

FIG. 8 shows an exemplary computer system for implementing the invention.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

Reference will now be made in detail to the preferred embodiments of the present invention, examples of which are illustrated in the accompanying drawings.

The present invention provides dynamic allocation storage space, required for supporting virtual storage in the Virtual Execution Environments. The following definitions are generally used throughout this description:

VEE—a type of environment that supports program code execution, where at least a part of the real hardware and software required for running program code are presented as their virtual analogs. From the point of view or the user, that the code in VEE runs as if it were running on the real computing system.

VPS—Virtual Private Server (or VE), is one type of a Virtual Execution Environment (VEE) running on the same hardware system with a shared OS kernel and most of the system resources, where isolation of Virtual Execution Environments is implemented on the namespace level. A Virtual Private Server (VPS) is a closed set, or collection, of processes, system resources, users, groups of users, objects and data structures. Each VPS has an ID, or some other identifier, that distinguishes it from other VPSs. The VPS offers to its users a service that is functionally substantially equivalent to a standalone server with remote access. From the perspective of an administrator of the VPS, the VPS should preferably act the same as a dedicated computer at a data center. For example, it is desirable for the administrator of the VPS to have the same remote access to the server through the Internet, the same ability to reload the server, load system and application software, authorize VPS users, establish disk space quotas of the users and user groups, support storage area networks (SANs), set up and configure network connections and web servers, etc. In other words, the full range of

6

system administrator functions is desirable, as if the VPS were a dedicated remote server, with the existence of the VPS being transparent from the perspective of both the VPS user and the VPS administrator.

VM—a type of an isolated Virtual Execution Environments running on the same physical machine simultaneously. Each Virtual Machine instance executes its own OS kernel. Support of Virtual Machines is implemented using a Virtual Machine Monitor and/or a Hypervisor. An example of a VM is a VMware Virtual Machine.

Hypervisor—control software having the highest privilege level for administrating hardware computer resources and Virtual Machines. One embodiment of the hypervisor is used in the Xen open source project.

Virtual storage—block-level storage space that may be regarded by the user of a computer system as addressable hardware storage, or a storage partition, using virtual addresses, that are utilized during virtual disk input/output operations as physical addresses.

Generic file or consistent set of files—represents storage device use of a disk image. Examples of generic files are VMware virtual disk and VPS private area. A generic file may have internal structures, for example, B+tree structures for providing update of disk image during I/O operation. One example of updating a generic file includes use of transactions. In one embodiment of the invention, the transactions are performed with direct access to the storage device, without using a file system cache normally used by the operating system for caching file system reads and writes.

Disk image—a set of data that represents contents of a disk storage or contents of a partition corresponding to virtual storage on a block level.

Virtual disk driver—an OS driver that enables other programs to interact with a virtual hardware device. The virtual disk driver may be implemented as a special driver or may be implemented by adding, to the standard hardware driver, additional functionality, for example, filtering or redirecting ability.

Free space—disk storage blocks that are mapped to the disk storage, but do not contain useful data. In other words, free blocks that are not used by the file system driver and their content is not generally useful. One embodiment of the invention provides a possibility of writing, to the generic file, only blocks that are used on the virtual image.

Migration—process of transferring data associated with VPS or VM using generic file.

Optionally, a generic file may contain checkpoints, which provide for migrating not only of the current state of the Virtual Execution Environment, but also of previous states. Other options include copying the VEE as a file to the disk storage, and exporting the VEE to another hardware system or cloning the VEE on the same computer.

Online snapshotting of an image—process of creating a disk snapshot without terminating current operations with the disk image. Since the disk image is stored as a file, snapshotting of the image at any predefined point in time may be implemented. All that is needed is creating additional file for registering transactions. Another embodiment includes copy-on-write snapshotting technique.

Transactions—the main requirement is secure handling of disk images. Registration of transactions is implemented as units of interaction of Virtual Execution Environments with virtual data storage. Transaction registration, along with creation generic files with proper structure, simplifies this. Emergency termination of registering transactions should not corrupt the disk image. It also simplifies creating snapshots of

US 8,539,137 B1

7

virtual storage. All virtual storage transactions are treated independently from each other.

Snapshot—data that correspond to all used blocks in virtual storage at a certain moment in time. It optionally may contain page file blocks and other runtime information.

The proposed approach is fault tolerant and makes it possible to improve performance of computing systems running VEEs, since more effective structures are used for storage.

Scalability and portability of the storage device is provided, since storage resources are dynamically allocated to VEE and are independent of the hardware storage configuration.

Disk images may be created on the storage drive (including hard drive or net storage drives), and might not contain free blocks, except for some number of virtual free blocks or data related to virtual storage drive capacity or capacity of free blocks, for example, the number of virtual clusters that are free or alignment data. Disk images have a predefined virtual storage capacity that is preferably constant for all VEEs, but may be changed with the help of system tools. Disk image size at system start is normally less than virtual storage capacity. Each disk image is accessible to user processes of at least one isolated Virtual Execution Environment (keeping in mind common access to virtual storages from VM or VPS, where, with real hardware, it is possible to share disk storages of different VMs, by providing access rights and organizing communication channels, for example, through emulating network communication.

The Virtual Execution Environment, as discussed herein, may be of any number of types. For example, they can be Virtual Private Servers. They can be Virtual Machines, such as those from VM ware or Microsoft. They can be Virtual Machines, in the model of XEN, that use a Hypervisor and a Service OS. They can also be Virtual Machines that utilize a Lightweight Hypervisor, such as those marketed by Parallels Software International. Also, SWsoft, Inc.'s isolated VEs may be used as another example.

FIG. 1 illustrates an embodiment where virtual drives are handled using virtual storage redirectors between Virtual Execution Environments and a low-level operating system that has direct access to computing hardware resources. Here, a disk driver of Virtual Execution Environment 1 (**128A**) may use a file system that is at least partially shared for all VEEs of its type, and therefore is redirected to a common disk driver that directly interfaces with the storage **102**. On the other hand, VEE 2 (**128B**) is redirected to a file system, and the OS **104** manages disk access.

The disk driver of Virtual Execution Environment 2 (**128B**) handles dedicated real files and therefore is redirected to the common file system.

FIG. 2 shows an embodiment of the invention wherein virtual drive supporting means are responsible for consistency of disk image files and use common file system for handling virtual disk reads and writes.

FIG. 3 shows an embodiment of a system where Guest Operating Systems use Primary Operating Systems drivers for access to storage drive. Herein Primary OS has direct access to computing system hardware. The present invention may be used in such a system.

FIG. 4 shows an embodiment where Guest Operating Systems use Primary Operating Systems drivers for access to storage drive while handling virtual storage. Here, Primary OS accesses computing system hardware via a common safe interface (a Hypervisor). Furthermore, Guest Operating Systems can access computing system hardware via the Hypervisor. This architecture is also available for using the present invention.

8

FIG. 5 shows an embodiment where virtual drive supporting means forms file structures corresponding to the virtual image storage in a safe manner with high performance.

For this purpose virtual drive supporting means assign areas of the storage device required for storing and writing virtual disk data. Also, virtual drive supporting means forms a file with an internal structure that includes user data storage blocks and redirection blocks. All the data formed by virtual drive supporting means are written to storage device using a file system driver and/or a device driver. In one embodiment of the invention, the file containing redirection blocks can include redirection blocks with a multilevel hierarchy, for example, it can have a B-tree or B+tree structure.

FIG. 1 illustrates one embodiment of the present invention. As shown in FIG. 1, in a system that has a number of Virtual Execution Environments (only two are shown in this figure, designated by **120A** and **120B**, although it should be understood that the actual number can be anywhere from one to hundreds or even thousands), and with a low level operating system **104**. The computer **802** has local storage **102**. The local storage **102** is typically a real (non-virtualized) hard drive, although other means for providing storage can be used, such as RAID5, network storage drives, SANs, etc. As noted earlier, the most common example today is a local hard drive.

Low-level operating system **104** has a file system **106**, and a disk driver **108** that it uses to actually “talk” to the local storage **102**. Data of the disk image may be stored directly inside the disk partition, which will require partition manager intervention. A partition manager **110** is most commonly activated at initialization, as is known conventionally. Each of the Virtual Execution Environments **120** can be running a user process, or multiple user processes, designated in FIG. 1 by **122A**, **122B**. Each Virtual Execution Environment also normally has its own file system, designated by **124A**, **124B** in FIG. 1. The file system **124** is virtualized—in other words, it is a file image that is stored in the local storage **102** in some fashion, as discussed further below. Each Virtual Execution Environment **120** also has a virtual disk driver, here, **126A**, **126B**. The virtual disk driver **126** is used to talk to virtualized storage (see **210** in FIG. 2) (although typically the Virtual Execution Environment itself is not aware that the storage **210** is virtualized). Actual interaction with the storage **102** is through a virtual storage redirector, labeled **128A**, **128B**.

The present invention makes it possible to overcommit storage to a particular Virtual Execution Environment, through the use of the mechanisms described below and illustrated in the figures. One of the facts of life of modern storage utilization is that frequently, many of the Virtual Execution Environments do not use all of the storage that is actually allocated to them—for example, many users want the ability to store large files, for example, video files, or large picture files, but, in practice, rarely take advantage of the full amount of storage available to them. Therefore, the virtualized file system **124**, illustrated in FIG. 1, can be virtualized as a file whose size can dynamically change, as needed.

FIG. 2 provides additional illustration of one embodiment of the present invention. As shown in FIG. 2, one of the Virtual Execution Environments **120B** running on the computer (in this case, a Virtual Machine-type VEE) has a user process **122B**, a file system **124B**, and a virtual disk driver **126B**. Virtual disk driver **126B** communicates to a redirection layer **202**. The redirection layer **202** interfaces to virtual image storage **210**, either directly or through a block request redirection layer **204**. This communication therefore involves two types of entities—user data, stored in user data storage blocks **208**, and block redirection data **206**. The block redirection

US 8,539,137 B1

9

data **206** can be viewed as a form of overhead, which the redirection layer **202** requires in order to figure out where the user data for this particular user is actually stored on the physical storage device **102**. Virtual image storage **210** content is stored as a file or a set of files on the physical storage device **102** and may be accessed by means of driver, which supports the file system **106**. Overhead data can contain, for example, information—where in the file is stored data related to partition offset in virtual storage.

The degree of control by the Primary OS (POS) **303** and the VMM **305** over processor resources and the processor's privileged instructions can vary. As examples, a hosted VMM system may be used, where VMM is installed on a host that runs an operating system independently of the VMM. FIG. 3, illustrates one option, such as provided by Parallels Software International, Inc. that uses a particular mechanism called the "Lightweight Hypervisor" **307** which restricts the ability of both the Primary OS **303A**, **303B** and the VMM **305A-305D** to issue instructions that affect the interrupt descriptor table (IDT). Thus, the Lightweight Hypervisor processes all interrupts and exceptions in the system and dispatches them to the POS and VMMs based on its virtualization policy.

Hypervisor-based VMMs combine the advantages of the prior art systems, and eliminate major disadvantages. The Hypervisor runs on the system level and creates Virtual Machines on the user level. One of the VMs runs a so-called "Primary OS" (POS) **303A** and has privileges to handle some of the hardware (e.g., CPU **309**, network interface card **311**) directly. Other VMs and the Hypervisor **307** uses Primary OS **303** and its hardware drivers for communication with the actual hardware. At the same time, the Hypervisor **307** employs efficient memory management, processor scheduling and resource management without the help of the Primary OS **303**. The advantages of the Hypervisor-based approach are high VM isolation, security, efficient resource management and a small trusted Hypervisor footprint.

The Lightweight Hypervisor **307** runs on the system level, reloads the Interrupt Descriptor Table (IDT) and protects it from modification by the Primary OS **303** and the VMM **305**. The Primary OS **303** and the VMMs **305** are not allowed to modify IDT and they are not allowed to independently process interrupts. The Lightweight Hypervisor **307** coexists in all address spaces (Primary OS' context and VMMs' contexts) and exclusively processes all hardware interrupts and exceptions. It is responsible for interrupt forwarding, context switching between the Primary OS **303** and VMMs **305**, and for efficient resource scheduling.

Thus, the Lightweight Hypervisor **307** is a small piece of software that helps manage VMMs **305** by handling hardware resource interrupts events. The Lightweight Hypervisor **307** captures the primary IDT pointer and replaces it with its own.

FIG. 4 illustrates another type of Virtual Execution Environment, here, the XEN type of virtualization. As shown in FIG. 4, many of the elements are similar to what is shown in FIG. 3, however, this type of system uses Service OS for direct access to the hardware, handling I/O and other types of interrupts.

FIG. 5 is another illustration of the embodiment of the present invention. Essentially, FIG. 5 is an alternative view of what is shown in FIGS. 1 and 2. As shown in FIG. 5, the virtual storage redirector **128** includes an I/O interface **502**, and a virtual drive supporting means **504**. The I/O interface **502** is what the operating system expects to see when it accesses a device, such as storage. The virtual drive supporting means **504** handles the overhead data **506** that is associated with storing the disk image in an optimal form.

10

In one embodiment, the optimal form of the disk image file may include specific file organization where disk image files are dynamically updated during writes to the virtual disk drive. In one embodiment, the disk image files contains only used blocks data, or the volume of "useless" data stored in the disk image files can be minimized. This is different from the conventional disk storages, where contents of unused blocks are stored on the disk unless the contents of unused blocks has been overwritten. To implement this feature, the disk image files can be reduced in size constantly or periodically to exclude data corresponding to unused blocks of the virtual disk drive. In one embodiment, unused blocks reflected in at least one image file may be overwritten by current write requests.

Examples of block redirection data **206**, shown in FIG. 2, although there could be other types of overhead data as discussed above. Also, user data **508** is the data that the user process **122** actually works with, and stores on the local storage **102**. The virtual storage redirector **128** communicates with the virtual image storage **210**, use block redirection data **206** when redirection is required and stores the redirection data **206** and the user data **208** on the storage **210**.

It should be noted that the virtual drive supporting means can run in operating system space and can be shared among different VEEs. As an example of implementation, each VEE's processes can have a unique ID corresponding to the VEE and the virtual drive supporting means can use a VEE-specific databases and settings for supporting I/O requests issued by processes of certain VEEs. Herein, a VEE-specific file system driver or even other drivers of the computing system may be implemented as a single driver that is common for multiple VEEs, but having VEE-specific behavior. In other words, each VEE has its own file system driver object or objects for handling a disk image of its VEE. Alternatively, different drivers for different VEEs can be used.

In some embodiments, the disk image can be created that includes files common to different VEEs. This disk image can be handled by using full access writes by processes of one VEE or by processes of the Host OS, and other VEEs can have read-only access to image data. Where multiple VEEs have a large amount of identical data, storing a single instance of identical data as a shared structure can save a lot of storage space. The identical shared data can be either the whole image or some disk image files or data related to files stored in the image.

FIG. 6 illustrates an internal structure of the file and time diagram of changing the structure the file when adding data to the file. FIG. 7 shows changing of the file structure of FIG. 6 during deletion of blocks of the virtual drive in the file and using those blocks for writing new data. See generally J. Gray and A. Reuter, "Transaction Processing: Techniques and Concepts", Morgan Kaufmann Pages 159-168 (1993); J. Gray and A. Reuter, "Transaction Processing: Techniques and Concepts", Morgan Kaufmann Pages 851-861 (1993); Mark Allen Weiss, Data Structures and Problem Solving Using C++ (Second Edition), Addison-Wesley, Pages 707-715 (2000), all incorporated herein by reference.

With reference to FIG. 8, an exemplary system for implementing the invention includes a general purpose computing device in the form of a personal computer or server **20** or the like, including a processing unit **21**, a system memory **22**, and a system bus **23** that couples various system components including the system memory to the processing unit **21**. The system bus **23** may be any of several types of bus structures including a memory bus or memory controller, a peripheral bus, and a local bus using any of a variety of bus architectures. The system memory includes read-only memory (ROM) **24**

US 8,539,137 B1

11

and random access memory (RAM) 25. A basic input/output system 26 (BIOS), containing the basic routines that help to transfer information between elements within the personal computer 20, such as during start-up, is stored in ROM 24. The personal computer 20 may further include a hard disk drive 27 for reading from and writing to a hard disk, not shown, a magnetic disk drive 28 for reading from or writing to a removable magnetic disk 29, and an optical disk drive 30 for reading from or writing to a removable optical disk 31 such as a CD-ROM, DVD-ROM or other optical media. The hard disk drive 27, magnetic disk drive 28, and optical disk drive 30 are connected to the system bus 23 by a hard disk drive interface 32, a magnetic disk drive interface 33, and an optical drive interface 34, respectively. The drives and their associated computer-readable media provide non-volatile storage of computer readable instructions, data structures, program modules and other data for the personal computer 20. Although the exemplary environment described herein employs a hard disk, a removable magnetic disk 29 and a removable optical disk 31, it should be appreciated by those skilled in the art that other types of computer readable media that can store data that is accessible by a computer, such as magnetic cassettes, flash memory cards, digital video disks, Bernoulli cartridges, random access memories (RAMs), read-only memories (ROMs) and the like may also be used in the exemplary operating environment.

A number of program modules may be stored on the hard disk, magnetic disk 29, optical disk 31, ROM 24 or RAM 25, including an operating system 35 (preferably Windows™ 2000). The computer 20 includes a file system 36 associated with or included within the operating system 35, such as the Windows NT™ File System (NTFS), one or more application programs 37, other program modules 38 and program data 39. A user may enter commands and information into the personal computer 20 through input devices such as a keyboard 40 and pointing device 42. Other input devices (not shown) may include a microphone, joystick, game pad, satellite dish, scanner or the like. These and other input devices are often connected to the processing unit 21 through a serial port interface 46 that is coupled to the system bus, but may be connected by other interfaces, such as a parallel port, game port or universal serial bus (USB). A monitor 47 or other type of display device is also connected to the system bus 23 via an interface, such as a video adapter 48. In addition to the monitor 47, personal computers typically include other peripheral output devices (not shown), such as speakers and printers.

The personal computer 20 may operate in a networked environment using logical connections to one or more remote computers 49. The remote computer (or computers) 49 may be another personal computer, a server, a router, a network PC, a peer device or other common network node, and typically includes many or all of the elements described above relative to the personal computer 20, although only a memory storage device 50 has been illustrated. The logical connections include a local area network (LAN) 51 and a wide area network (WAN) 52. Such networking environments are commonplace in offices, enterprise-wide computer networks, Intranets and the Internet.

When used in a LAN networking environment, the personal computer 20 is connected to the local network 51 through a network interface or adapter 53. When used in a WAN networking environment, the personal computer 20 typically includes a modem 54 or other means for establishing communications over the wide area network 52, such as the Internet. The modem 54, which may be internal or external, is connected to the system bus 23 via the serial port interface 46. In a networked environment, program modules

12

depicted relative to the personal computer 20, or portions thereof, may be stored in the remote memory storage device. It will be appreciated that the network connections shown are exemplary and other means of establishing a communications link between the computers may be used.

Having thus described a preferred embodiment, it should be apparent to those skilled in the art that certain advantages of the described method and apparatus have been achieved. It should also be appreciated that various modifications, adaptations, and alternative embodiments thereof may be made within the scope and spirit of the present invention. The invention is further defined by the following claims.

What is claimed is:

1. A method for storing data of a Virtual Execution environment (VEE), comprising:

launching, on a computer system, isolated VEEs with a shared OS kernel providing services to the VEEs in response to requests from the VEEs;

starting a common storage device driver and a common file system driver;

mounting a virtual disk drive for each VEE, wherein each VEE has its own file system driver object for handling a disk image of its VEE;

wherein the virtual disk drive is represented on the a storage device as a disk image, and

wherein the disk image is stored on a physical storage device or on a network storage device as at least one file; and

wherein the file of the virtual disk drive comprises an internal structure visible to the shared OS kernel that includes user data storage blocks and redirection blocks that point to user data storage blocks and that are used by the VEE's file system driver, and

wherein at least some of the virtual drives permit an over-commitment of disk space to their corresponding VEEs relative to standard allocation of disk space.

2. The method of claim 1, wherein the file is a generic file whose size is dynamically changed when content of the virtual disk drive is changed.

3. The method of claim 1, wherein the at least one file has a B-tree or B+tree structure.

4. The method of claim 1, wherein the redirection blocks have a multi-level hierarchy.

5. The method of claim 1, wherein the VEEs' file system drivers and the common driver supporting the virtual disk drive are running in operating system space and are used by different VEEs.

6. The method of claim 1, wherein the disk image further comprises files common to different VEEs.

7. The method of claim 1, wherein the disk image files are dynamically updated during writes to the virtual disk drive.

8. The method of claim 1, wherein the disk image files are reduced in size to exclude data corresponding to unused blocks of the virtual disk drive.

9. The method of claim 1, wherein the disk image files contain only used blocks data.

10. The method of claim 1, wherein some disk image files can be shared between VEEs.

11. The method of claim 1, wherein the drive image files support starting the VEE contents on another hardware system or other VEE.

12. The method of claim 1, wherein an internal structure of the at least one file that allows for writes to the disk image in a fault tolerant manner; and

any virtual disk writes are added to the at least one file with preserving the internal structure integrity in any moment of time.

US 8,539,137 B1

13

13. The method of claim 1, further comprising online snapshotting of the virtual disk drive using an internal structure of the at least one file.

14. The method of claim 1, further comprising offline snapshotting of the virtual disk drive using an internal structure of the at least one file.

15. A method for storing data of a Virtual Machine, comprising:

starting a host operating system running a computing system;

starting a Virtual Machine Monitor (VMM) under control of the host operating system,

wherein the VMM virtualizes the computing system and has privileges as high as the host operating system;

creating isolated Virtual Machines (VMs), running on the computing system simultaneously, wherein each VM executes its own guest OS kernel and each VM runs under the control of the VMM;

starting a storage device driver and a file system driver in the host operating system;

mounting a virtual disk drive;

starting VM-specific file system drivers in the VMs, wherein the VM specific file system driver together with common storage device drivers support the virtual disk drive,

wherein the virtual disk drive is represented on a storage device as a disk image,

wherein the disk image data are stored on the storage device as at least one file that includes user data storage blocks and redirection blocks visible to the host operating system, the redirection blocks point to user data storage blocks,

wherein the redirection blocks have a multilevel hierarchy, and

an internal structure of the at least one file is used by the VM-specific file system driver,

wherein at least some of VMs utilize virtual free blocks as a part of disk image to permit an over-commitment of disk space.

16. The method of claim 15, wherein each VM uses a common storage device driver and a common file system driver.

17. A method for storing data of a Virtual Machine, comprising:

starting a Hypervisor that has exclusive control over at least some system resources and an interrupt descriptor table;

starting a Virtual Machine Monitor (VMM) running with system level privileges and which is co-resident in system space with a host operating system (HOS), wherein the Hypervisor allocates resources to the VMM;

creating a primary Virtual Machine (VM) without system level privileges and relocating the HOS to the primary VM, the deprived HOS having direct access to at least some I/O devices;

creating a secondary Virtual Machine running without system level privileges;

starting a storage device driver and a file system driver in the HOS;

starting a VM-specific file system driver in the secondary VM;

starting a VM-specific virtual device driver in the secondary VM that provides access to the virtual storage drive, wherein the VM specific virtual device driver has access to disk image files from a virtual device driver via the file system driver and a storage device driver; and

mounting a virtual disk drive,

14

wherein the virtual disk drive is represented on a storage device as a disk image, and

wherein the disk image data is stored on the storage device as at least one file; and

wherein the disk image comprises an internal structure that includes user data storage

blocks and redirection blocks that point to user data storage blocks and that are used by the VM-specific file system driver,

wherein the user data storage blocks and the redirection blocks of the virtual disk drive are visible to the host operating system, and

wherein the host OS, the primary VM and the secondary VM access hardware of the computing system via a common secure interface.

18. A method for storing data of Virtual Execution Environments, comprising:

starting a Virtual Machine Monitor (VMM) with system level privileges;

creating a primary Virtual Machine (VM) without system level privileges and relocating a host operating system (HOS) to the primary VM;

starting a safe interface providing secure access from the VMs to computing system hardware;

creating a secondary Virtual Machine running without system level privilege, wherein the secondary VM uses the safe interface for accessing hardware;

starting a storage device driver and a file system driver in the HOS;

starting a VM-specific file system driver in the secondary VM;

starting a VM-specific virtual device driver in the secondary VM that provides access to a virtual storage drive and has access to disk image files from the VM-specific virtual device driver via the file system driver, the storage device driver and the safe interface; and

mounting a virtual disk drive for the secondary VM;

wherein the virtual disk drive is represented on a storage device as a disk image; and

the disk image data is stored on the storage device as at least one file; and

wherein the disk image comprises an internal structure that includes user data storage blocks and redirection blocks that point to user data storage blocks and that are used by the VM-specific file system driver,

wherein the user data storage blocks and the redirection blocks of the virtual disk drive are visible to the host operating system, and

wherein at least some of the data of the VMs is accessible by multiple VMs, and access attempts by the VMs to other VM's data are redirected to the host OS.

19. A method for storing data of Virtual Execution Environments, comprising:

starting a host operating system on a computing system that uses a system level file system cache for caching data being read or written to a storage device;

starting a Virtual Machine Monitor (VMM) under the control of the host operating system;

starting isolated Virtual Machines (VMs) on the computing system, wherein each VM executes its own guest OS kernel and each VM runs under the control of the VMM;

starting a common storage device driver and a common file system driver in the operating system;

mounting a virtual disk drive, the virtual disk drive provides correct functioning of the VMs;

US 8,539,137 B1

15

starting VM-specific file system drivers in the VMs that, jointly with the common driver, supports a virtual disk drive that is represented on a storage device as a disk image;
wherein the disk image data is stored on the storage device as at least one file;
wherein writes to the at least one file are executed with bypassing the file system cache;
wherein an internal structure of the file enables writes to the disk image in a fault tolerant manner; and
wherein any virtual disk writes are added to the file while preserving the internal structure integrity of the file at any moment of time; and
wherein the internal structure of the file includes user data storage blocks and redirection blocks that are visible to the host OS and that point to user data storage blocks and that are used by the VM-specific file system driver, and wherein at least some files in the disk image are write-accessible by the host operating system and by one VM while being only read-accessible by other VMs.
20. A method for storing data of Virtual Private Servers (VPSs), comprising:
starting an operating system on a computing system;
initiating a plurality of VPSs on the computing system, all the VPSs sharing a single instance of the operating system that provides services to the VPSs in response to requests from the VPSs;
allocating, to each VPS, a dynamic virtual partition, wherein the size of the dynamic virtual partition changes in real time as VPS storage requirements change; and

16

wherein the dynamic virtual partition contains used and unused data blocks that are visible to the operating system; and
wherein at least some of the virtual partitions permit an over-commitment of resources to their corresponding VEEs relative to standard allocation of resources.
21. A computer program product for storing data of Virtual Machines (VMs), the computer program product comprising a non-transitory computer useable medium having computer program logic stored thereon for executing on at least one processor, the computer program logic comprising:
computer program code means for starting a host operating system on a computing system;
computer program code means for initiating a Hypervisor having full system privileges;
computer program code means for initiating a plurality of VMs on a single OS, the VMs being managed by a Virtual Machine Monitor (VMM),
wherein at least one of the VMs is running a deprivileged Service OS, and
wherein the Hypervisor has higher privileges than the VMM, processes all interrupts, protects the Interrupt Descriptor Table from the host operating system and handles resource allocation; and
computer program code means for generating a dynamic virtual partition allocated to each VM, wherein the size of the dynamic virtual partition changes in real time as VM storage requirements change, and
wherein the dynamic virtual partition contains used and unused data blocks that are visible to the host operating system.

* * * * *

EXHIBIT G

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re application of:

EMELYANOV *et al.*

Appl. No.: 14251989

Filed: April 14, 2014

For: **METHOD FOR TARGETED
RESOURCE VIRTUALIZATION IN
CONTAINERS**

Confirmation No.: 7069

Art Unit: 2196

Examiner: KESSLER, GREGORY AARON

Atty. Docket: 2230.1400000

Amendment and Reply Under 37 C.F.R. § 1.111

Mail Stop Amendment

Commissioner for Patents
PO Box 1450
Alexandria, VA 22313-1450

Sir:

In reply to the Office Action dated **February 24, 2016**, Applicants submit the following Amendment and Remarks. This Amendment is provided in the following format:

- (A) Each section begins on a separate sheet;
- (B) Starting on a separate sheet, a complete listing of all of the claims;
- (C) Starting on a separate sheet, the Remarks.

It is not believed that extensions of time or fees for net addition of claims are required beyond those that may otherwise be provided for in documents accompanying this paper.

However, if additional extensions of time are necessary to prevent abandonment of this application, then such extensions of time are hereby petitioned under 37 C.F.R. § 1.136(a), and any fees required therefor (including fees for net addition of claims) are hereby authorized to be charged to our Deposit Account No. 50-3523.

Amendments to the Claims

The listing of claims will replace all prior versions, and listings of claims in the application.

1. (currently amended) A system for targeted virtualization in containers, the system comprising:

a host hardware node having a host OS kernel;

a plurality of host OS kernel objects implemented on the host hardware node; and

at least one container running on the host hardware node, the container virtualizing the host OS and using selected host OS kernel utilities,

wherein:

the host OS kernel utilities have a virtualization on-off switch;

the selected host OS kernel object is virtualized inside the container if the utility virtualization switch is on; and

the container uses the host OS kernel objects shared with other containers running on the hardware node.

2. (original) The system of claim 1, wherein the utility virtualization switch is turned on by a container administrator.

3. (original) The system of claim 1, wherein the utility virtualization switch is turned on based on a container user requirements.

4. (original) The system of claim 1, wherein the host OS kernel objects are not virtualized.

5. (original) The system of claim 1, wherein the host OS kernel objects are shared among containers running on the hardware node.

6. (original) The system of claim 1, wherein the container uses selected virtualized host OS kernel objects and shared host OS kernel objects based on user requirements.

7. (original) The system of claim 1, wherein the host OS kernel is patched for selected virtualization of utilities in a form of separate utility modules.

8. (original) The system of claim 1, wherein the host OS kernel objects used for selected virtualization are any of:

memory;

I/O operations;

disk;

network;

users and groups;

devices;

PID tree;

IPC objects;

user applications; and

system modules.

9. (original) The system of claim 1, wherein the container is a part of a cloud-based infrastructure.

10. (currently amended) A computer-implemented method for targeted virtualization in a container, the method comprising:

- (a) launching an OS kernel on a host hardware node;
- (b) patching the host OS kernel for selected virtualization of host OS kernel objects;
- (c) activating a utility virtualization on-off switch on the host OS kernel objects;
- (d) selecting the host OS kernel objects to be virtualized by turning the utility virtualization switch on;
- (e) launching a first container on the hardware node, the first container virtualizing the host OS;
- (f) virtualizing the selected host OS kernel objects inside the container;
- (g) repeating steps (b) – (f) for another container launched on the hardware node, wherein the first container shares the host OS kernel objects that are not virtualized with other containers running on the hardware node.

11. (original) The method of claim 10, wherein the selecting of the host OS kernel objects is implemented based on user requirements.

12. (original) The method of claim 10, wherein the containers cannot access objects virtualized inside the other containers.

13. (original) The method of claim 10, wherein the containers form a cloud infrastructure.

14. (original) The method of claim 10, further comprising creating a beancounter for controlling resource usage by container processes.

15. (original) The method of claim 14, wherein virtualized container resources are controlled by the beancounter.

16. (currently amended) A computer-implemented method for backup optimization in a dedicated container, the method comprising:

- (a) launching an OS kernel on a host hardware node;
- (b) patching the host OS kernel for selected virtualization of host OS kernel objects;
- (c) activating a utility virtualization on-off switch on the host OS kernel objects;
- (d) selecting the host OS kernel objects to be virtualized by turning the utility

virtualization switch on;

(e) launching a dedicated backup container on the hardware node, the backup container virtualizing the host OS;

- (f) virtualizing the selected host OS kernel objects inside the dedicated container; and
- (g) starting a host backup utility configured to backup container data,

wherein the host backup utility only backs up the host OS kernel objects that are not virtualized within the dedicated backup container.

17. (original) A system for targeted virtualization in a container, the system comprising:

a processor;

a memory coupled to the processor; and

a computer program logic stored in the memory and executed on the processor, the computer program logic for implementing the steps (a) – (g) of claim 10.

- 6 -

EMELYANOV *et al.*
Appl. No. 14251989

Remarks

Reconsideration of this Application is respectfully requested.

Upon entry of the foregoing amendment, claims 1-17 are pending in the application.

These changes are believed to introduce no new matter, and their entry is respectfully requested.

Based on the above amendment and the following remarks, Applicants respectfully request that the Examiner reconsider all outstanding objections and rejections and that they be withdrawn.

Interview

Applicants thank the Examiner for the courtesies extended during the interview. As discussed during the interview, the amendments are intended to highlight the fact that the present claims are directed to different form of virtualization – containers that virtualize the host OS – rather than to VM-type virtualization, that virtualizes the processor.

Rejections under 35 U.S.C. § 103

Claims 1-6, 8 and 9 are rejected under 35 U.S.C. 103 as being unpatentable over Applicant-Admitted Prior Art (hereinafter AAPA) in view of Ross et al (U.S. Pat. Pub. No. 2005/0091652). These rejections are respectfully traversed.

Essentially, the difference between Ross and what is claimed is that the claimed **container virtualizes the host OS** (which is why there is only one host OS, and no guest OS's in container-based systems), while a **Virtual Machine**, in Ross, **virtualizes the processor** (which is why a VM has a guest OS). These are two different (and unrelated) forms of virtualization.

To that end, the Office Action argues:

Atty. Dkt. No. 2230.1400000

- 7 -

EMELYANOV *et al.*
Appl. No. 14251989

However, Ross teaches that the host OS kernel utilities have a virtualization on/off switch and the selected host OS kernel object is virtualized inside the container if the utility virtualization switch is on (Paragraph [0049], Lines 1-8 teaches a virtualization on/off switch including virtualization and non-virtualization modes).

Respectfully, this is incorrect. Para. 0049 describes hardware virtualization aspects of the Intel processor, and is relevant only to VMs – not to containers. There are no containers in Ross, and Ross does not virtualize host OS kernel objects – not inside containers (or inside VMs), not outside them. Note that the amended claim language clarifies that the container virtualizes the host OS (and not the processor) – making Ross inapplicable.

Also, it is incorrect to say that “host OS kernel utilities have a virtualization on/off switch”. Ross has a virtualization on/off switch – but it does not virtualize host OS kernel utilities. Note that a canonical VM is unaware of the existence of the host OS (regardless of any hardware virtualization on/off switches).

Reconsideration is respectfully requested.

Conclusion

All of the stated grounds of objection and rejection have been properly traversed, accommodated, or rendered moot. Applicants therefore respectfully request that the Examiner reconsider all presently outstanding objections and rejections and that they be withdrawn. Applicants believe that a full and complete reply has been made to the outstanding Office Action and, as such, the present application is in condition for allowance. If the Examiner believes, for any reason, that personal communication will expedite prosecution of this application, the Examiner is invited to telephone the undersigned at the number provided.

Atty. Dkt. No. 2230.1400000

- 8 -

EMELYANOV *et al.*
Appl. No. 14251989

Prompt and favorable consideration of this Amendment and Reply is respectfully
requested.

Respectfully submitted,

BARDMESSER LAW GROUP

/GB/

George S. Bardmesser
Attorney for Applicants
Registration No. 44,020

Date: April 4, 2016

1025 Connecticut Avenue, N.W., Suite 1000
Washington, D.C. 20006
(202) 293-1191

Electronic Acknowledgement Receipt

EFS ID:	25387548
Application Number:	14251989
International Application Number:	
Confirmation Number:	7069
Title of Invention:	METHOD FOR TARGETED RESOURCE VIRTUALIZATION IN CONTAINERS
First Named Inventor/Applicant Name:	PAVEL EMELIANOV
Customer Number:	54089
Filer:	George Simon Bardmesser
Filer Authorized By:	
Attorney Docket Number:	2230.1400000
Receipt Date:	04-APR-2016
Filing Date:	14-APR-2014
Time Stamp:	14:37:14
Application Type:	Utility under 35 USC 111(a)

Payment information:

Submitted with Payment	no
------------------------	----

File Listing:

Document Number	Document Description	File Name	File Size(Bytes)/ Message Digest	Multi Part /.zip	Pages (if appl.)
1		AMENDMENTAFTERFIRSTOFFICE ACTION.pdf	46743 572310dd48bedec629bfe32a2c320ee60d8911c	yes	8

Multipart Description/PDF files in .zip description

	Multipart Description/PDF files in .zip description		
	Document Description	Start	End
	Amendment/Req. Reconsideration-After Non-Final Reject	1	1
	Claims	2	5
	Applicant Arguments/Remarks Made in an Amendment	6	8

Warnings:**Information:****Total Files Size (in bytes):**

46743

This Acknowledgement Receipt evidences receipt on the noted date by the USPTO of the indicated documents, characterized by the applicant, and including page counts, where applicable. It serves as evidence of receipt similar to a Post Card, as described in MPEP 503.

New Applications Under 35 U.S.C. 111

If a new application is being filed and the application includes the necessary components for a filing date (see 37 CFR 1.53(b)-(d) and MPEP 506), a Filing Receipt (37 CFR 1.54) will be issued in due course and the date shown on this Acknowledgement Receipt will establish the filing date of the application.

National Stage of an International Application under 35 U.S.C. 371

If a timely submission to enter the national stage of an international application is compliant with the conditions of 35 U.S.C. 371 and other applicable requirements a Form PCT/DO/EO/903 indicating acceptance of the application as a national stage submission under 35 U.S.C. 371 will be issued in addition to the Filing Receipt, in due course.

New International Application Filed with the USPTO as a Receiving Office

If a new international application is being filed and the international application includes the necessary components for an international filing date (see PCT Article 11 and MPEP 1810), a Notification of the International Application Number and of the International Filing Date (Form PCT/RO/105) will be issued in due course, subject to prescriptions concerning national security, and the date shown on this Acknowledgement Receipt will establish the international filing date of the application.

PATENT APPLICATION FEE DETERMINATION RECORD Substitute for Form PTO-875					Application or Docket Number 14/251,989		Filing Date 04/14/2014		<input type="checkbox"/> To be Mailed		
ENTITY: <input checked="" type="checkbox"/> LARGE <input type="checkbox"/> SMALL <input type="checkbox"/> MICRO											
APPLICATION AS FILED – PART I											
(Column 1)			(Column 2)								
FOR		NUMBER FILED		NUMBER EXTRA		RATE (\$)		FEE (\$)			
<input type="checkbox"/> BASIC FEE (37 CFR 1.16(a), (b), or (c))		N/A		N/A		N/A					
<input type="checkbox"/> SEARCH FEE (37 CFR 1.16(k), (i), or (m))		N/A		N/A		N/A					
<input type="checkbox"/> EXAMINATION FEE (37 CFR 1.16(o), (p), or (q))		N/A		N/A		N/A					
TOTAL CLAIMS (37 CFR 1.16(i))		minus 20 =		*		X \$ =					
INDEPENDENT CLAIMS (37 CFR 1.16(h))		minus 3 =		*		X \$ =					
<input type="checkbox"/> APPLICATION SIZE FEE (37 CFR 1.16(s))		If the specification and drawings exceed 100 sheets of paper, the application size fee due is \$310 (\$155 for small entity) for each additional 50 sheets or fraction thereof. See 35 U.S.C. 41(a)(1)(G) and 37 CFR 1.16(s).									
<input type="checkbox"/> MULTIPLE DEPENDENT CLAIM PRESENT (37 CFR 1.16(j))											
* If the difference in column 1 is less than zero, enter "0" in column 2.						TOTAL					
APPLICATION AS AMENDED – PART II											
(Column 1)			(Column 2)			(Column 3)					
AMENDMENT	04/04/2016		CLAIMS REMAINING AFTER AMENDMENT			HIGHEST NUMBER PREVIOUSLY PAID FOR		PRESENT EXTRA			
	Total (37 CFR 1.16(i))		* 17		Minus	** 20		= 0			
	Independent (37 CFR 1.16(h))		* 3		Minus	*** 3		= 0			
	<input type="checkbox"/> Application Size Fee (37 CFR 1.16(s))										
	<input type="checkbox"/> FIRST PRESENTATION OF MULTIPLE DEPENDENT CLAIM (37 CFR 1.16(j))										
						TOTAL ADD'L FEE		0			
(Column 1)			(Column 2)			(Column 3)					
AMENDMENT			CLAIMS REMAINING AFTER AMENDMENT			HIGHEST NUMBER PREVIOUSLY PAID FOR		PRESENT EXTRA			
	Total (37 CFR 1.16(i))		*		Minus	**		=			
	Independent (37 CFR 1.16(h))		*		Minus	***		=			
	<input type="checkbox"/> Application Size Fee (37 CFR 1.16(s))										
	<input type="checkbox"/> FIRST PRESENTATION OF MULTIPLE DEPENDENT CLAIM (37 CFR 1.16(j))										
						TOTAL ADD'L FEE					
<p>* If the entry in column 1 is less than the entry in column 2, write "0" in column 3.</p> <p>** If the "Highest Number Previously Paid For" IN THIS SPACE is less than 20, enter "20".</p> <p>*** If the "Highest Number Previously Paid For" IN THIS SPACE is less than 3, enter "3".</p> <p>The "Highest Number Previously Paid For" (Total or Independent) is the highest number found in the appropriate box in column 1.</p>											

LIE
/MARSHA RICHARDS/

This collection of information is required by 37 CFR 1.16. The information is required to obtain or retain a benefit by the public which is to file (and by the USPTO to process) an application. Confidentiality is governed by 35 U.S.C. 122 and 37 CFR 1.14. This collection is estimated to take 12 minutes to complete, including gathering, preparing, and submitting the completed application form to the USPTO. Time will vary depending upon the individual case. Any comments on the amount of time you require to complete this form and/or suggestions for reducing this burden, should be sent to the Chief Information Officer, U.S. Patent and Trademark Office, U.S. Department of Commerce, P.O. Box 1450, Alexandria, VA 22313-1450. DO NOT SEND FEES OR COMPLETED FORMS TO THIS ADDRESS. **SEND TO: Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450.**

If you need assistance in completing the form, call 1-800-PTO-9199 and select option 2.

EXHIBIT H

- 28 -

WHAT IS CLAIMED IS:

1. A server comprising:
a host running an operating system (OS) kernel;
a plurality of isolated virtual private servers (VPSs) supported within the operating system kernel;
an application available to users of the VPSs; and
an interface giving the users access to the application.
2. The server of claim 1, wherein each VPS has its own set of addresses.
3. The server of claim 2, wherein each VPS has its own objects.
4. The server of claim 3, wherein each object has a unique identifier in a context of the operating system kernel.
5. The server of claim 1, wherein each VPS cannot affect a process of another VPS.
6. The server of claim 1, wherein each VPS cannot affect threads of a process of another VPS.
7. The server of claim 1, wherein each VPS cannot affect an object of another VPS.
8. The server of claim 1, wherein each VPS cannot affect an object of the OS kernel.

- 29 -

9. The server of claim 1, wherein each VPS cannot access information about a process running on another VPS.

10. The server of claim 1, wherein each VPS includes:
isolation of address space of a user of one VPS from address space of a user on another VPS;
isolation of server resources for each VPS; and
isolation of application failure effects.

11. The server of claim 1, wherein the host includes any of:
a virtual memory allocated to each user;
a pageable memory used by the OS kernel and by user processes;
physical memory used by the user processes;
objects and data structures used by the OS kernel;
I/O resources;
file space; and
individual user resource limitations.

12. The server of claim 10, wherein each VPS includes:
a plurality of processes and threads servicing corresponding users;
a plurality of objects associated with the plurality of processes and threads;
a set of unique user IDs corresponding to users of a particular VPS;
a unique file space;
means for management of the particular VPS;
means for management of services offered by the particular VPS to its users; and

- 30 -

means for delivery of the services to the users of the particular VPS.

13. The server of claim 1, wherein the server includes any of the following:

- a capability of allocating a resource to a designated VPS;
- a capability of reallocating the resource to a designated VPS;
- a capability of allocating the resource to a VPS in current need of resources;
- a capability of reallocating the resource to a VPS in current need of resources;
- a capability of dynamically reallocating the resource from one VPS to another VPS when this resource is available;
- a capability of dynamically reallocating the resource from one VPS to another VPS when commanded by the OS kernel; and
- a capability of compensating a particular VPS in a later time slice for under-use or over-use of the resource by the particular VPS in a current time slice.

14. The server of claim 13, wherein the server defines time slices, such that the resource is allocated for each time slice.

15. The server of claim 13, wherein the server defines time slices, such that the resource is reallocated for each time slice from one VPS to any of another VPS, the OS kernel, an application software daemon and a system software daemon.

16. The server of claim 15, wherein the server dynamically partitions and dedicates the resource to the VPSs based on a service level agreement.

- 31 -

17. The server of claim 1, wherein all the VPSs are supported within the same OS kernel.

18. The server of claim 1, wherein some functionality of the VPSs is supported in user mode.

19. The server of claim 1, wherein the server is implemented as an add-on to any of Microsoft Windows NT server, Microsoft Windows 2000 server, and Microsoft Windows Server 2003 server.

20. The server of claim 1, wherein the server is implemented as an add-on to a server based on a Microsoft Windows product.

21. The server of claim 1, wherein the operating system kernel includes at least one process and thread for processing of user requests.

22. A server comprising:
a computer system running an operating system (OS) kernel;
a plurality of virtual private servers (VPSs) running on the computer system, wherein each VPS functionally appears to a user as a dedicated server; and
a plurality of applications running within the VPSs and available to users of the VPSs.

23. The server of claim 22, wherein each VPS is isolated from any other VPS.

24. The server of claim 22, wherein each VPS has its own set of addresses.

25. The server of claim 24, wherein each VPS has its own objects.

- 32 -

26. The server of claim 25, wherein each object has a unique identifier in a context of the OS kernel.

27. The server of claim 22, wherein each VPS cannot affect a process of another VPS.

28. The server of claim 22, wherein each VPS cannot affect an object of another VPS.

29. The server of claim 22, wherein each VPS cannot access information about a process running on another VPS.

30. The server of claim 22, wherein each VPS includes:
isolation of a set of addresses of a user of one VPS from addresses of a user of another VPS;
isolation of server resources for each VPS; and
isolation of application program failure effects.

31. The server of claim 22, wherein the server includes any of the following resources:
a virtual memory allocated to each user;
a pageable memory used by the OS kernel and by user processes;
physical memory used by the user processes;
objects and data structures used by the OS kernel;
I/O resources;
file space; and
individual user resource limitations.

- 33 -

32. The server of claim 22, wherein each VPS includes the following resources:

- a plurality of processes and threads servicing corresponding users;

- a plurality of objects associated with the plurality of processes and threads;

- a set of unique user IDs corresponding to users of a particular VPS;

- a unique file space;

- means for management of the particular VPS;

- means for management of services offered by the particular VPS to its users; and

- means for delivery of the services to the users of the particular VPS.

33. The server of claim 22, wherein the server includes any of the following capabilities:

- a capability of allocating a server resource to a designated VPS;

- a capability of reallocating the server resource to a designated VPS;

- a capability of allocating the server resource to a VPS in current need of that resource;

- a capability of reallocating the server resource to a VPS in current need of that resource;

- a capability of dynamically reallocating the server resource from one VPS to another VPS when that server resource is available;

- a capability of dynamically reallocating the server resource from one VPS to another VPS when commanded by the OS kernel; and

- a capability of compensating a particular VPS in a later time slice for under-use or over-use of the server resource by the particular VPS in a current time slice.

- 34 -

34. The server of claim 33, wherein the server defines time slices, such that the server resource is allocated for each time slice.

35. The server of claim 22, wherein the server dynamically partitions and dedicates server resources to the VPSs based on a service level agreement.

36. The server of claim 22, wherein all the VPSs are supported within the same OS kernel.

37. The server of claim 22, wherein some functionality of the VPSs is supported in user mode.

38. The server of claim 22, wherein the server is implemented as an add-on to any of Microsoft Windows NT server, Microsoft Windows 2000 server, and Microsoft Windows Server 2003 server.

39. The server of claim 22, wherein the server is implemented as an add-on to a server based on a Microsoft Windows product.

40. The server of claim 22, wherein the operating system includes at least one process and thread for execution of user requests.

41. A server comprising:
a host running an operating system (OS) kernel; and
a plurality of isolated virtual private servers (VPSs) running on the host,

wherein the server includes a capability of dynamically reallocating host resources to a first VPS when a second VPS is under-utilizing its resources.

- 35 -

42. The server of claim 41, wherein the server includes a capability of compensating a particular VPS in a later period for under-use or over-use of the host resources by the particular VPS in a current period.

43. A method of managing a server comprising:
defining a virtual private server (VPS) ID corresponding to a VPS;
receiving a request from a VPS process to create an object;
creating an internal operating system (OS) kernel representation of the object;
checking whether such an object already exists in OS kernel storage;
if no such object exists in the OS kernel storage, creating an instance of the object to be associated with the VPS ID; and
if such an object already exists in the OS kernel storage, one of rejecting the request and returning the existing object to the VPS process.

44. The method of claim 43, wherein the VPS process is a user mode process.

45. The method of claim 43, wherein the VPS process is an application running within the VPS.

46. The method of claim 43, further including the step of storing a representation of the newly created instance of the object in the OS kernel storage.

47. The method of claim 46, wherein the storage is an OS kernel cache.

- 36 -

48. The method of claim 46, wherein the storage is a data structure in OS kernel memory.

49. A method of managing server resources comprising:
creating a plurality of isolated virtual private servers (VPSs) running on a host;
allocating host resources to each VPS based on a corresponding service level agreement (SLA); and
dynamically changing the host resources available to a particular VPS when such host resources are available.

50. The method of claim 49, wherein the host resources allocated to the particular VPS are available due to underutilization by other VPSs.

51. The method of claim 49, wherein the host resources include any of CPU usage, disk space, physical memory, virtual memory and bandwidth.

52. The method of claim 49, further including the step of reserving a particular host resource for the particular VPS.

53. The method of claim 49, wherein the step of dynamically changing the host resources allocates resources that exceed SLA guarantees.

54. The method of claim 54, wherein the step of dynamically changing the host resources is performed when the SLA specifies a soft upper limit on resource allocation.

55. The method of claim 49, further including the step of compensating the particular VPS in one time period for host resource underavailability in an earlier time period.

- 37 -

56. A method of managing a server comprising:
creating operating system (OS) kernel structures that handle isolated virtual private server (VPS) specific objects;
initiating instances of VPSs with corresponding VPS IDs, wherein all the VPS instances are supported within the OS kernel;
generating instances of OS kernel objects corresponding to the VPS IDs; and
providing services to users of the VPSs.

57. The method of claim 56, further including the step of allocating resources to the VPSs.

58. The method of claim 57, further including the step of dynamically adjusting the allocated resources for each VPS.

59. The method of claim 56, further including the step of terminating a VPS and all its associated objects and processes based on the VPS ID.

60. A system for managing a server comprising:
means for defining a virtual private server (VPS) ID;
means for receiving a request from a VPS process to create an object;
means for creating an internal operating system (OS) kernel representation of the object;
means for checking whether such an object already exists in OS kernel storage;
means for creating an instance of the object to be associated with the VPS ID if no such object exists in the OS kernel storage; and

- 38 -

means for rejecting the request if such an object already exists in the OS kernel storage.

61. The system of claim 60, wherein the VPS process is a user mode process.

62. The system of claim 60, wherein each VPS appears to a user as functionally equivalent to a remotely accessible server.

63. The system of claim 60, wherein the VPS process is an application running within the VPS.

64. The system of claim 60, further including means for storing a representation of the object in the OS kernel storage.

65. The system of claim 60, wherein the storage is a data structure in OS kernel memory.

66. A system for managing server resources comprising:
means for creating a plurality of isolated virtual private servers (VPSs) running on a host;
means for allocating host resources to each VPS based on its service level agreement (SLA); and
means for dynamically changing the host resources available to a particular VPS when such host resources are available.

67. The system of claim 66, wherein the host resources allocated to the particular VPS are available due to underutilization by other VPSs.

68. The system of claim 66, wherein the host resources include any of CPU usage, disk space, physical memory, virtual memory and bandwidth.

- 39 -

69. The system of claim 66, further including means for reserving a particular host resource for the particular VPS.

70. The system of claim 66, wherein the means for dynamically changing the host resources allocates resources that exceed SLA guarantees to the particular VPS.

71. The system of claim 70, wherein the means for dynamically changing the host resources increases the host resources when the SLA specifies a soft upper limit on resource allocation.

72. The system of claim 66, further including means for compensating the particular VPS in one time period for host resource underavailability in an earlier time period.

73. The system of claim 66, wherein each VPS appears to a user as functionally equivalent to a remotely accessible server.

74. A system for managing a server comprising:
means for creating operating system (OS) kernel structures that handle virtual private server (VPS) specific objects;
means for initiating instances of VPSs with corresponding IDs, wherein all the instances are supported within the OS kernel;
means for generating instances of OS kernel objects corresponding to the VPS IDs; and
means for providing services to users of the VPSs.

75. The system of claim 74, further including means for dynamically adjusting resources allocated to each VPS.

- 40 -

76. The system of claim 74, further including means for terminating a VPS and all its associated objects and processes based on the VPS ID.

77. The system of claim 74, wherein each VPS appears to a user as functionally equivalent to a remotely accessible server.

78. A computer program product for managing a server, the computer program product comprising a computer useable medium having computer program logic recorded thereon for controlling a processor, the computer program logic comprising:

computer program code means for defining a virtual private server (VPS) ID;

computer program code means for receiving a request from a VPS process to create an object;

computer program code means for creating an internal operating system (OS) kernel representation of the object;

computer program code means for checking whether such an object already exists in OS kernel storage;

computer program code means for creating a new instance of the object to be associated with the VPS ID if no such object exists in the OS kernel storage; and

computer program code means for rejecting the request if such an object already exists in the OS kernel storage.

79. A computer program product for managing server resources, the computer program product comprising a computer useable medium having computer program logic recorded thereon for controlling a processor, the computer program logic comprising:

computer program code means for creating a plurality of isolated virtual private servers (VPSs) running on a host;

- 41 -

computer program code means for allocating host resources to each VPS based on a corresponding service level agreement; and

computer program code means for dynamically changing host resources available to a particular VPS when such host resources are available.

80. A computer program product for managing a server, the computer program product comprising a computer useable medium having computer program logic recorded thereon for controlling a processor, the computer program logic comprising:

computer program code means for creating operating system (OS) kernel structures that handle virtual private server (VPS) specific objects;

computer program code means for initiating instances of VPSs with corresponding IDs, wherein all the instances are supported within the OS kernel;

computer program code means for generating instances of OS kernel objects corresponding to the VPS IDs; and

computer program code means for providing services to users of the VPSs.



UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
 United States Patent and Trademark Office
 Address: COMMISSIONER FOR PATENTS
 P.O. Box 1450
 Alexandria, Virginia 22313-1450
 www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
10/703,594	11/10/2003	Serguei M. Belousov	2230.0010003/MBR/GSB	2218
54089 7590 05/05/2008 BARDESSER LAW GROUP, P.C. 910 17TH STREET, N.W. SUITE 800 WASHINGTON, DC 20006			EXAMINER POLLACK, MELVIN H	
			ART UNIT	PAPER NUMBER
			2145	
			MAIL DATE	DELIVERY MODE
			05/05/2008	PAPER

Please find below and/or attached an Office communication concerning this application or proceeding.

The time period for reply, if any, is set in the attached communication.

Interview Summary	Application No. 10/703,594	Applicant(s) BELOUSSOV ET AL.	
	Examiner MELVIN H. POLLACK	Art Unit 2145	

All participants (applicant, applicant's representative, PTO personnel):

(1) MELVIN H. POLLACK. (3) ____.

(2) George Bardmesser (44,020). (4) ____.

Date of Interview: 02 May 2008.

Type: a) ☐ Telephonic b) ☐ Video Conference
c) ☒ Personal [copy given to: 1) ☐ applicant 2) ☐ applicant's representative]

Exhibit shown or demonstration conducted: d) ☐ Yes e) ☒ No.
If Yes, brief description: _____.

Claim(s) discussed: 1 and 5-13.

Identification of prior art discussed: Huang et al. (7,219,354), Kauffman (6,633,916)(6,332,180).

Agreement with respect to the claims f) ☒ was reached. g) ☐ was not reached. h) ☐ N/A.

Substance of Interview including description of the general nature of what was agreed to if an agreement was reached, or any other comments: See Continuation Sheet.

(A fuller description, if necessary, and a copy of the amendments which the examiner agreed would render the claims allowable, if available, must be attached. Also, where no copy of the amendments that would render the claims allowable is available, a summary thereof must be attached.)

THE FORMAL WRITTEN REPLY TO THE LAST OFFICE ACTION MUST INCLUDE THE SUBSTANCE OF THE INTERVIEW. (See MPEP Section 713.04). If a reply to the last Office action has already been filed, APPLICANT IS GIVEN A NON-EXTENDABLE PERIOD OF THE LONGER OF ONE MONTH OR THIRTY DAYS FROM THIS INTERVIEW DATE, OR THE MAILING DATE OF THIS INTERVIEW SUMMARY FORM, WHICHEVER IS LATER, TO FILE A STATEMENT OF THE SUBSTANCE OF THE INTERVIEW. See Summary of Record of Interview requirements on reverse side or on attached sheet.

/Melvin H Pollack/
 Examiner, Art Unit 2145

 Examiner's signature, if required

Examiner Note: You must sign this form unless it is an Attachment to a signed Office action.

Manual of Patent Examining Procedure (MPEP), Section 713.04, Substance of Interview Must be Made of Record

A complete written statement as to the substance of any face-to-face, video conference, or telephone interview with regard to an application must be made of record in the application whether or not an agreement with the examiner was reached at the interview.

Title 37 Code of Federal Regulations (CFR) § 1.133 Interviews
Paragraph (b)

In every instance where reconsideration is requested in view of an interview with an examiner, a complete written statement of the reasons presented at the interview as warranting favorable action must be filed by the applicant. An interview does not remove the necessity for reply to Office action as specified in §§ 1.111, 1.135. (35 U.S.C. 132)

37 CFR §1.2 Business to be transacted in writing.

All business with the Patent or Trademark Office should be transacted in writing. The personal attendance of applicants or their attorneys or agents at the Patent and Trademark Office is unnecessary. The action of the Patent and Trademark Office will be based exclusively on the written record in the Office. No attention will be paid to any alleged oral promise, stipulation, or understanding in relation to which there is disagreement or doubt.

The action of the Patent and Trademark Office cannot be based exclusively on the written record in the Office if that record is itself incomplete through the failure to record the substance of interviews.

It is the responsibility of the applicant or the attorney or agent to make the substance of an interview of record in the application file, unless the examiner indicates he or she will do so. It is the examiner's responsibility to see that such a record is made and to correct material inaccuracies which bear directly on the question of patentability.

Examiners must complete an Interview Summary Form for each interview held where a matter of substance has been discussed during the interview by checking the appropriate boxes and filling in the blanks. Discussions regarding only procedural matters, directed solely to restriction requirements for which interview recordation is otherwise provided for in Section 812.01 of the Manual of Patent Examining Procedure, or pointing out typographical errors or unreadable script in Office actions or the like, are excluded from the interview recordation procedures below. Where the substance of an interview is completely recorded in an Examiners Amendment, no separate Interview Summary Record is required.

The Interview Summary Form shall be given an appropriate Paper No., placed in the right hand portion of the file, and listed on the "Contents" section of the file wrapper. In a personal interview, a duplicate of the Form is given to the applicant (or attorney or agent) at the conclusion of the interview. In the case of a telephone or video-conference interview, the copy is mailed to the applicant's correspondence address either with or prior to the next official communication. If additional correspondence from the examiner is not likely before an allowance or if other circumstances dictate, the Form should be mailed promptly after the interview rather than with the next official communication.

The Form provides for recordation of the following information:

- Application Number (Series Code and Serial Number)
- Name of applicant
- Name of examiner
- Date of interview
- Type of interview (telephonic, video-conference, or personal)
- Name of participant(s) (applicant, attorney or agent, examiner, other PTO personnel, etc.)
- An indication whether or not an exhibit was shown or a demonstration conducted
- An identification of the specific prior art discussed
- An indication whether an agreement was reached and if so, a description of the general nature of the agreement (may be by attachment of a copy of amendments or claims agreed as being allowable). Note: Agreement as to allowability is tentative and does not restrict further action by the examiner to the contrary.
- The signature of the examiner who conducted the interview (if Form is not an attachment to a signed Office action)

It is desirable that the examiner orally remind the applicant of his or her obligation to record the substance of the interview of each case. It should be noted, however, that the Interview Summary Form will not normally be considered a complete and proper recordation of the interview unless it includes, or is supplemented by the applicant or the examiner to include, all of the applicable items required below concerning the substance of the interview.

A complete and proper recordation of the substance of any interview should include at least the following applicable items:

- 1) A brief description of the nature of any exhibit shown or any demonstration conducted,
- 2) an identification of the claims discussed,
- 3) an identification of the specific prior art discussed,
- 4) an identification of the principal proposed amendments of a substantive nature discussed, unless these are already described on the Interview Summary Form completed by the Examiner,
- 5) a brief identification of the general thrust of the principal arguments presented to the examiner,
(The identification of arguments need not be lengthy or elaborate. A verbatim or highly detailed description of the arguments is not required. The identification of the arguments is sufficient if the general nature or thrust of the principal arguments made to the examiner can be understood in the context of the application file. Of course, the applicant may desire to emphasize and fully describe those arguments which he or she feels were or might be persuasive to the examiner.)
- 6) a general indication of any other pertinent matters discussed, and
- 7) if appropriate, the general results or outcome of the interview unless already described in the Interview Summary Form completed by the examiner.

Examiners are expected to carefully review the applicant's record of the substance of an interview. If the record is not complete and accurate, the examiner will give the applicant an extendable one month time period to correct the record.

Examiner to Check for Accuracy

If the claims are allowable for other reasons of record, the examiner should send a letter setting forth the examiner's version of the statement attributed to him or her. If the record is complete and accurate, the examiner should place the indication, "Interview Record OK" on the paper recording the substance of the interview along with the date and the examiner's initials.

Continuation of Substance of Interview including description of the general nature of what was agreed to if an agreement was reached, or any other comments: Applicant desires to move away from virtual machines (i.e. VMWare) to lower memory and CPU requirements, increase scalability, and isolate a logical server's resource impact on other machines (i.e. one virtual private server cannot use all the CPU cycles at the expense of the other VPS).

Applicant agreed to amend to all independent claims that there is a single instance of an Operating System, which includes a kernel to control the virtual servers, such that multiple virtual servers share a set of OS files. Applicant also agreed to amend the claims to disclose a scheduler located on the kernel to allocate CPU cycles so that no VPS may monopolize the physical system's resources, as distinct from a partitioner or dynamic allocator. As a backup, applicant will also amend claim 13's compensation limitation to stress that the compensation of resources happens at a time after the under or overuse of a resource, and to place in a separate dependent claim.

Examiner agrees that the new claims read over the cited art, and will allow the case, barring an updated search with a focus on scheduler, shadow volumes and compensation.

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re application of:

Belousov *et al.*

Appl. No.: 10/703,594

Filed: November 10, 2003

For: **Virtual Private Server with Isolation of
System Components**

Confirmation No.: 2218

Art Unit: 2145

Examiner: POLLACK, MELVIN H.

Atty. Docket: 2230.0010003/MBR/GSB

Amendment and Reply Under 37 C.F.R. § 1.111

Mail Stop Amendment

Commissioner for Patents
PO Box 1450
Alexandria, VA 22313-1450

Sir:

In reply to the Office Action dated **April 11, 2008**, Applicants submit the following Amendment and Remarks. This Amendment is provided in the following format:

- (A) Each section begins on a separate sheet;
- (B) Starting on a separate sheet, amendments to the specification by presenting replacement paragraphs marked up to show changes made;
- (C) Starting on a separate sheet, a complete listing of all of the claims:
 - in ascending order;
 - with status identifiers; and
 - with markings in the currently amended claims;
- (D) Starting on a separate sheet, the Remarks.

It is not believed that extensions of time or fees for net addition of claims are required beyond those that may otherwise be provided for in documents accompanying this paper.

However, if additional extensions of time are necessary to prevent abandonment of this

- 2 -

Belousov *et al.*
Appl. No. 10/703,594

application, then such extensions of time are hereby petitioned under 37 C.F.R. § 1.136(a), and any fees required therefor (including fees for net addition of claims) are hereby authorized to be charged to our Deposit Account No. 50-3523.

Amendments to the Abstract

Please amend the abstract to read as follows:

A server includes a host running an operating system kernel. Isolated virtual private servers (VPSs) are supported within the kernel. At least one application is available to users of the VPS. A plurality of interfaces give the users access to the application. Each VPS has its own set of addresses. Each object of each VPS has a unique identifier in a context of the operating system kernel. Each VPS is isolated from objects and processes of another VPS. Each VPS includes isolation of address space of each user from address space of a user on any other VPS, isolation of server resources for each VPS, and failure isolation. The server includes a capability of allocating (or reallocating) system resources to a designated VPS, allocating (or reallocating) system resources to a VPS in current need of such resources, dynamically allocating (or reallocating) VPS resources to a VPS when additional resources are available, and compensating a particular VPS in a later period for a period of under-use or over-use of server resources by the particular VPS in a current period. VPS resources are allocated for each time cycle. ~~The server dynamically partitions and dedicates resources to the VPSs based on a service level agreement.~~ All the VPSs are supported within the same OS kernel. ~~Some functionality of the VPSs can be supported in user mode. The server is implemented as an add-on to a server based on a Microsoft Windows product, and the VPSs appear to the users as dedicated servers.~~

Amendments to the Claims

The listing of claims will replace all prior versions, and listings of claims in the application.

1. (currently amended) A server comprising:
 - a host running ~~an~~ a single operating system (OS) kernel;
 - a plurality of isolated virtual private servers (VPSs) supported within the operating system kernel and all the VPSs sharing the same single operating system kernel;
 - a user space/kernel space interface that virtualizes the OS kernel and includes a CPU time scheduler that allocates CPU time between threads of the VPSs and ensures that a thread of one VPS does not adversely affect a CPU time allocation of threads of other VPSs;
 - an application available to users of the VPSs; and
 - an application interface that includes system calls, shared memory interfaces, and I/O driver control (ioctl)s for giving the users access to the application.
2. (currently amended) The server of claim 1, wherein each VPS has its own set of addresses and its own set of objects.
3. (currently amended) The server of claim ~~[[2]]~~ 1, wherein ~~each VPS has its own objects~~ the scheduler ensures that a thread of one VPS does not adversely affect a CPU time allocation of threads of other VPSs by setting guaranteed levels of CPU time usage for each user process and/or VPS.
4. (original) The server of claim 3, wherein each object has a unique identifier in a context of the operating system kernel.
5. (currently amended) The server of claim 1, wherein the server includes a capability of compensating a particular VPS in a later time slice for under-use or over-use of the

resource by the particular VPS in a current time slice ~~each VPS cannot affect a process of another VPS.~~

6. (currently amended) The server of claim 1, ~~wherein each VPS cannot affect threads of a process of another VPS~~ further comprising isolation of server resources for each VPS.

7. (original) The server of claim 1, wherein each VPS cannot affect an object of another VPS.

8. (original) The server of claim 1, wherein each VPS cannot affect an object of the OS kernel.

9. (original) The server of claim 1, wherein each VPS cannot access information about a process running on another VPS.

10. (currently amended) The server of claim 1, wherein each VPS includes:
isolation of address space of a user of one VPS from address space of a user on another VPS;
~~isolation of server resources for each VPS;~~ and
isolation of application failure effects.

11. (currently amended) The server of claim 1, wherein the host includes any of:
a virtual memory allocated to each user;
a pageable memory used by the OS kernel and by user processes;
physical memory used by the user processes;
objects and data structures used by the OS kernel;
I/O resources; and
file space; ~~and~~

~~individual user resource limitations.~~

12. (original) The server of claim 10, wherein each VPS includes:
 - a plurality of processes and threads servicing corresponding users;
 - a plurality of objects associated with the plurality of processes and threads;
 - a set of unique user IDs corresponding to users of a particular VPS;
 - a unique file space;
 - means for management of the particular VPS;
 - means for management of services offered by the particular VPS to its users; and
 - means for delivery of the services to the users of the particular VPS.

13. (currently amended) The server of claim 1, wherein the server includes any of the following:
 - a capability of allocating a resource to a designated VPS;
 - a capability of reallocating the resource to a designated VPS;
 - a capability of allocating the resource to a VPS in current need of resources;
 - a capability of reallocating the resource to a VPS in current need of resources;
 - a capability of dynamically reallocating the resource from one VPS to another VPS when this resource is available; and
 - a capability of dynamically reallocating the resource from one VPS to another VPS when commanded by the OS kernel; ~~and~~
 - ~~a capability of compensating a particular VPS in a later time slice for under-use or over-use of the resource by the particular VPS in a current time slice.~~

14. (currently amended) The server of claim [[13]] 6, wherein the server defines time slices, such that the resource is allocated for each time slice.

15. (currently amended) The server of claim [[13]] 6, wherein the server defines time slices, such that the resource is reallocated for each time slice from one VPS to any of another VPS, the OS kernel, an application software daemon and a system software daemon.

16. (original) The server of claim 15, wherein the server dynamically partitions and dedicates the resource to the VPSs based on a service level agreement.

17. (original) The server of claim 1, wherein all the VPSs are supported within the same OS kernel.

18. (original) The server of claim 1, wherein some functionality of the VPSs is supported in user mode.

19. (original) The server of claim 1, wherein the server is implemented as an add-on to any of Microsoft Windows NT server, Microsoft Windows 2000 server, and Microsoft Windows Server 2003 server.

20. (original) The server of claim 1, wherein the server is implemented as an add-on to a server based on a Microsoft Windows product.

21. (original) The server of claim 1, wherein the operating system kernel includes at least one process and thread for processing of user requests.

22. (currently amended) A server comprising:
a computer system running ~~an~~ a single operating system (OS) kernel;
a plurality of virtual private servers (VPSs) running on the computer system,
wherein each VPS functionally appears to a user as a dedicated server, and all the VPSs sharing the same single operating system kernel;

a user space/kernel space interface that virtualizes the OS kernel and includes a CPU time scheduler that allocates CPU time between threads of the VPSs and prevents a thread of one VPS from adversely affecting a CPU time allocation of threads of other VPSs; and

a plurality of applications running within the VPSs and available to users of the VPSs; and

an application interface that includes system calls, shared memory interfaces, and I/O driver control (ioctl) for giving the users for giving the users access to the application.

23. (original) The server of claim 22, wherein each VPS is isolated from any other VPS.

24. (original) The server of claim 22, wherein each VPS has its own set of addresses.

25. (original) The server of claim 24, wherein each VPS has its own objects.

26. (original) The server of claim 25, wherein each object has a unique identifier in a context of the OS kernel.

27. (currently amended) The server of claim [[22]] 1, wherein each VPS manages individual user resource limitations ~~each VPS cannot affect a process of another VPS.~~

28. (original) The server of claim 22, wherein each VPS cannot affect an object of another VPS.

29. (original) The server of claim 22, wherein each VPS cannot access information about a process running on another VPS.

30. (original) The server of claim 22, wherein each VPS includes:

isolation of a set of addresses of a user of one VPS from addresses of a user of another VPS;

isolation of server resources for each VPS; and

isolation of application program failure effects.

31. (original) The server of claim 22, wherein the server includes any of the following resources:

a virtual memory allocated to each user;

a pageable memory used by the OS kernel and by user processes;

physical memory used by the user processes;

objects and data structures used by the OS kernel;

I/O resources;

file space; and

individual user resource limitations.

32. (original) The server of claim 22, wherein each VPS includes the following resources:

a plurality of processes and threads servicing corresponding users;

a plurality of objects associated with the plurality of processes and threads;

a set of unique user IDs corresponding to users of a particular VPS;

a unique file space;

means for management of the particular VPS;

means for management of services offered by the particular VPS to its users; and

means for delivery of the services to the users of the particular VPS.

33. (original) The server of claim 22, wherein the server includes any of the following capabilities:

a capability of allocating a server resource to a designated VPS;

a capability of reallocating the server resource to a designated VPS;

a capability of allocating the server resource to a VPS in current need of that resource;

a capability of reallocating the server resource to a VPS in current need of that resource;

a capability of dynamically reallocating the server resource from one VPS to another VPS when that server resource is available;

a capability of dynamically reallocating the server resource from one VPS to another VPS when commanded by the OS kernel; and

a capability of compensating a particular VPS in a later time slice for under-use or over-use of the server resource by the particular VPS in a current time slice.

34. (original) The server of claim 33, wherein the server defines time slices, such that the server resource is allocated for each time slice.

35. (original) The server of claim 22, wherein the server dynamically partitions and dedicates server resources to the VPSs based on a service level agreement.

36. (original) The server of claim 22, wherein all the VPSs are supported within the same OS kernel.

37. (original) The server of claim 22, wherein some functionality of the VPSs is supported in user mode.

38. (original) The server of claim 22, wherein the server is implemented as an add-on to any of Microsoft Windows NT server, Microsoft Windows 2000 server, and Microsoft Windows Server 2003 server.

39. (original) The server of claim 22, wherein the server is implemented as an add-on to a server based on a Microsoft Windows product.

40. (original) The server of claim 22, wherein the operating system includes at least one process and thread for execution of user requests.

41. (currently amended) A server comprising:
a host running ~~an~~ a single operating system (OS) kernel; ~~and~~
a plurality of isolated virtual private servers (VPSs) running on the host, and all the VPSs sharing the same single operating system kernel;
a user space/kernel space interface that virtualizes the OS kernel and includes a CPU time scheduler that allocates CPU time between threads of the VPSs and prevents a thread of one VPS from adversely affecting a CPU time allocation of threads of other VPSs;
wherein the server includes a capability of dynamically reallocating host resources to a first VPS when a second VPS is under-utilizing its resources; and
wherein the server includes an application interface that includes system calls, shared memory interfaces, and I/O driver control (ioctl) for giving the users access to the application.

42. (original) The server of claim 41, wherein the server includes a capability of compensating a particular VPS in a later period for under-use or over-use of the host resources by the particular VPS in a current period.

43. (currently amended) A method of managing a server comprising:
defining a virtual private server (VPS) ID corresponding to a VPS, wherein multiple VPSs are running on the server and all the VPSs share a single operating system kernel of the server;
wherein the server includes a user space/kernel space interface that virtualizes the operating system kernel and includes a CPU time scheduler that allocates CPU time between threads of the VPSs and prevents a thread of one VPS from adversely affecting a CPU time allocation of threads of other VPSs, and

wherein the server includes an application interface that includes system calls, shared memory interfaces, and I/O driver control (ioctl) for giving the users access to the application;

receiving a request from a VPS process to create an object;
creating an internal operating system (OS) kernel representation of the object;
checking whether such an object already exists in OS kernel storage;
if no such object exists in the OS kernel storage, creating an instance of the object to be associated with the VPS ID; and
if such an object already exists in the OS kernel storage, one of rejecting the request and returning the existing object to the VPS process.

44. (original) The method of claim 43, wherein the VPS process is a user mode process.

45. (original) The method of claim 43, wherein the VPS process is an application running within the VPS.

46. (original) The method of claim 43, further including the step of storing a representation of the newly created instance of the object in the OS kernel storage.

47. (original) The method of claim 46, wherein the storage is an OS kernel cache.

48. (original) The method of claim 46, wherein the storage is a data structure in OS kernel memory.

49. (currently amended) A method of managing server resources comprising:
creating a plurality of isolated virtual private servers (VPSs) running on a host
and all the VPSs sharing the same single operating system kernel running on the host;

the host including a user space/kernel space interface that virtualizes the OS kernel and includes a CPU time scheduler that allocates CPU time between threads of the VPSs and prevents a thread of one VPS from adversely affecting a CPU time allocation of threads of other VPSs, and

wherein the host includes an application interface that includes system calls, shared memory interfaces, and I/O driver control (ioctl) for giving the users access to the application;

allocating host resources to each VPS based on a corresponding service level agreement (SLA); and

dynamically changing the host resources available to a particular VPS when such host resources are available.

50. (original) The method of claim 49, wherein the host resources allocated to the particular VPS are available due to underutilization by other VPSs.

51. (original) The method of claim 49, wherein the host resources include any of CPU usage, disk space, physical memory, virtual memory and bandwidth.

52. (original) The method of claim 49, further including the step of reserving a particular host resource for the particular VPS.

53. (original) The method of claim 49, wherein the step of dynamically changing the host resources allocates resources that exceed SLA guarantees.

54. (original) The method of claim 54, wherein the step of dynamically changing the host resources is performed when the SLA specifies a soft upper limit on resource allocation.

55. (original) The method of claim 49, further including the step of compensating the particular VPS in one time period for host resource underavailability in an earlier time period.

56. (currently amended) A method of managing a server comprising:
creating structures of a single operating system (OS) kernel ~~structures~~ that handle isolated virtual private server (VPS) specific objects;
initiating instances of VPSs with corresponding VPS IDs, wherein all the VPS instances are supported within the OS kernel and all the VPSs share the same single operating system kernel;
wherein the host includes a user space/kernel space interface that virtualizes the OS kernel and includes a CPU time scheduler that allocates CPU time between threads of the VPSs and prevents a thread of one VPS from adversely affecting a CPU time allocation of threads of other VPSs, and
wherein the host includes an application interface that includes system calls, shared memory interfaces, and I/O driver control (ioctl) for giving the users access to the application;
generating instances of OS kernel objects corresponding to the VPS IDs; and
providing services to users of the VPSs.

57. (original) The method of claim 56, further including the step of allocating resources to the VPSs.

58. (original) The method of claim 57, further including the step of dynamically adjusting the allocated resources for each VPS.

59. (original) The method of claim 56, further including the step of terminating a VPS and all its associated objects and processes based on the VPS ID.

60. (currently amended) A system for managing a server comprising:
means for defining a virtual private server (VPS) ID, wherein multiple VPSs are running on the server and all the VPSs share a single operating system kernel of the server;

wherein the server includes a user space/kernel space interface that virtualizes the operating system kernel and includes a CPU time scheduler that allocates CPU time between threads of the VPSs and prevents a thread of one VPS from adversely affecting a CPU time allocation of threads of other VPSs, and

wherein the host includes an application interface that includes system calls, shared memory interfaces, and I/O driver control (ioctl) for giving the users access to the application;

means for receiving a request from a VPS process to create an object;

means for creating an internal operating system (OS) kernel representation of the object;

means for checking whether such an object already exists in OS kernel storage;

means for creating an instance of the object to be associated with the VPS ID if no such object exists in the OS kernel storage; and

means for rejecting the request if such an object already exists in the OS kernel storage.

61. (original) The system of claim 60, wherein the VPS process is a user mode process.

62. (original) The system of claim 60, wherein each VPS appears to a user as functionally equivalent to a remotely accessible server.

63. (original) The system of claim 60, wherein the VPS process is an application running within the VPS.

64. (original) The system of claim 60, further including means for storing a representation of the object in the OS kernel storage.

65. (original) The system of claim 60, wherein the storage is a data structure in OS kernel memory.

66. (currently amended) A system for managing server resources comprising:
means for creating a plurality of isolated virtual private servers (VPSs) running on a host, wherein multiple VPSs are running on the host and all the VPSs share a single operating system kernel of the server;

wherein the host includes a user space/kernel space interface that virtualizes the operating system kernel and includes a CPU time scheduler that allocates CPU time between threads of the VPSs and prevents a thread of one VPS from adversely affecting a CPU time allocation of threads of other VPSs, and

wherein the host includes an application interface that includes system calls, shared memory interfaces, and I/O driver control (ioctl) for giving the users access to the application;

means for allocating host resources to each VPS based on its service level agreement (SLA); and

means for dynamically changing the host resources available to a particular VPS when such host resources are available.

67. (original) The system of claim 66, wherein the host resources allocated to the particular VPS are available due to underutilization by other VPSs.

68. (original) The system of claim 66, wherein the host resources include any of CPU usage, disk space, physical memory, virtual memory and bandwidth.

69. (original) The system of claim 66, further including means for reserving a particular host resource for the particular VPS.

70. (original) The system of claim 66, wherein the means for dynamically changing the host resources allocates resources that exceed SLA guarantees to the particular VPS.

71. (original) The system of claim 70, wherein the means for dynamically changing the host resources increases the host resources when the SLA specifies a soft upper limit on resource allocation.

72. (original) The system of claim 66, further including means for compensating the particular VPS in one time period for host resource underavailability in an earlier time period.

73. (original) The system of claim 66, wherein each VPS appears to a user as functionally equivalent to a remotely accessible server.

74. (currently amended) A system for managing a server comprising:
means for creating structures of a single operating system (OS) kernel ~~structures~~ that handle virtual private server (VPS) specific objects;

means for initiating instances of VPSs with corresponding IDs, wherein all the instances are supported within the OS kernel, wherein all the VPSs share a single operating system kernel of the server;

wherein the server includes a user space/kernel space interface that virtualizes the operating system kernel and includes a CPU time scheduler that allocates CPU time between threads of the VPSs and prevents a thread of one VPS from adversely affecting a CPU time allocation of threads of other VPSs, and

wherein the server includes an application interface that includes system calls, shared memory interfaces, and I/O driver control (ioctl) for giving the users access to the application;

means for generating instances of OS kernel objects corresponding to the VPS IDs; and

means for providing services to users of the VPSs.

75. (original) The system of claim 74, further including means for dynamically adjusting resources allocated to each VPS.

76. (original) The system of claim 74, further including means for terminating a VPS and all its associated objects and processes based on the VPS ID.

77. (original) The system of claim 74, wherein each VPS appears to a user as functionally equivalent to a remotely accessible server.

78. (currently amended) A computer program product for managing a server, the computer program product comprising a computer useable storage medium having computer program logic recorded thereon for controlling a processor, the computer program logic comprising:

computer program code means for defining a virtual private server (VPS) ID, wherein multiple VPSs are running on the server and all the VPSs share a single operating system kernel of the server;

wherein the server includes a user space/kernel space interface that virtualizes the operating system kernel and includes a CPU time scheduler that allocates CPU time between threads of the VPSs and prevents a thread of one VPS from adversely affecting a CPU time allocation of threads of other VPSs, and

wherein the server includes an application interface that includes system calls, shared memory interfaces, and I/O driver control (ioctl) for giving the users access to the application;

computer program code means for receiving a request from a VPS process to create an object;

computer program code means for creating an internal operating system (OS) kernel representation of the object;

computer program code means for checking whether such an object already exists in OS kernel storage;

computer program code means for creating a new instance of the object to be associated with the VPS ID if no such object exists in the OS kernel storage; and

computer program code means for rejecting the request if such an object already exists in the OS kernel storage.

79. (currently amended) A computer program product for managing server resources, the computer program product comprising a computer useable storage medium having computer program logic recorded thereon for controlling a processor, the computer program logic comprising:

computer program code means for creating a plurality of isolated virtual private servers (VPSs) running on a host, wherein multiple VPSs are running on the host and all the VPSs share a single operating system kernel of the server;

wherein the host includes a user space/kernel space interface that virtualizes the operating system kernel and includes a CPU time scheduler that allocates CPU time between threads of the VPSs and prevents a thread of one VPS from adversely affecting a CPU time allocation of threads of other VPSs, and

wherein the host includes an application interface that includes system calls, shared memory interfaces, and I/O driver control (ioctl) for giving the users access to the application;

computer program code means for allocating host resources to each VPS based on a corresponding service level agreement; and

computer program code means for dynamically changing host resources available to a particular VPS when such host resources are available.

80. (currently amended) A computer program product for managing a server, the computer program product comprising a computer useable storage medium having computer program logic recorded thereon for controlling a processor, the computer program logic comprising:

computer program code means for creating operating system (OS) kernel structures that handle virtual private server (VPS) specific objects, wherein multiple VPSs are running on the host and all the VPSs share a single operating system kernel of the server;

wherein the host includes a user space/kernel space interface that virtualizes the operating system kernel and includes a CPU time scheduler that allocates CPU time between threads of the VPSs and prevents a thread of one VPS from adversely affecting a CPU time allocation of threads of other VPSs, and

wherein the host includes an application interface that includes system calls, shared memory interfaces, and I/O driver control (ioctl) for giving the users access to the application;

computer program code means for initiating instances of VPSs with corresponding IDs, wherein all the instances are supported within the OS kernel;

computer program code means for generating instances of OS kernel objects corresponding to the VPS IDs; and

computer program code means for providing services to users of the VPSs.

Remarks

Reconsideration of this Application is respectfully requested.

Upon entry of the foregoing amendment, claims 1-80 are pending in this application. Claims 1, 5, 6, 10, 13, 14, 15, 22, 27, 41, 43, 49, 56, 60, 66, 74, 78, 79 and 80 are amended. These changes are believed to introduce no new matter, and their entry is respectfully requested.

In the Office Action dated April 11, 2008, the abstract is objected to. Claims 78-80 stand rejected under 35 U.S.C. § 101 as being allegedly directed to non-statutory subject matter. Claims 1-12, 17-18, 21-32, 36-37, 40, 43-48, 56, 59-65, 74, 76-78 and 80 stand rejected under 35 U.S.C. § 102(e) as being allegedly anticipated by Huang *et al.*, U.S. Patent No. 7,219,354. Claims 13, 33, 41-42, 57-58 and 79 stand rejected under 35 U.S.C. § 103(a) as being allegedly unpatentable over Huang in view of Kauffman, U.S. Patent No. 6,633,916. Claims 14-15 and 34 stand rejected under 35 U.S.C. § 103(a) as being allegedly unpatentable over Huang and Kauffman in view of Gee *et al.*, U.S. Patent No. 6,374,286. Claims 16, 35, 49-55, 66-73 and 75 stand reject under 35 U.S.C. § 103(a) as being allegedly unpatentable over Huang, Kauffman and Gee in view of Keshav *et al.*, U.S. Patent No. 6,985,937. Claims 19-20, 38-39 stand rejected under 35 U.S.C. § 103(a) as being allegedly unpatentable over Huang in view of Lee, U.S. Patent No. 6,802,063.

Based on the above amendment and the following remarks, Applicants respectfully request that the Examiner reconsider all outstanding objections and rejections and that they be withdrawn.

Interview at the USPTO on May 2, 2008

Applicants' representative thanks the Examiner for the courtesies extended during the in person interview at the USPTO. All of the claims have been amended, as discussed during the interview. Applicants believe that, as discussed during the interview, these amendments overcome all the references of record, singly or in combination, as discussed further below.

Objection to the Abstract

A substitute abstract is submitted, to address the objection.

Rejections under 35 U.S.C. 101

The preambles of claims 78-80 are amended, to recite a computer useable storage medium. Applicants respectfully request that these rejections be withdrawn.

Rejections under 35 U.S.C. §§ 102(e) and 103(a)

All of the claims stand rejected based on Huang, or Huang in combination with other references. Although Applicants do not necessarily agree with the reasoning expressed in the Office Action, Applicants have amended the claims to recite several additional aspects. First, all of the independent claims have been amended to recite that only a single instance of an operating system is running on the server. This clearly distinguishes the claims over Kauffman, which has multiple operating systems running on a single machine.

Furthermore, all of the claims have been amended to recite that the host or server includes a user/kernel interface (which is sometimes referred to as a "kernel abstraction layer" in more modern terminology), and which includes a CPU time scheduler. Huang does not disclose a CPU time scheduler anywhere in his specification, and the logical conclusion is that Huang

uses a scheduler that is part of the standard operating system service/utility. Thus, this feature alone clearly distinguishes over Huang, or any combination of Huang with other references.

Additionally, all the claims now recite that the CPU time scheduler prevents a thread of one VPS from adversely affecting the thread of another VPS. This is inherent in the fact that the scheduler is part of the kernel abstraction layer, and therefore can manage the time allocated to the threads of the various VPSs. In Huang, due to the fact that the scheduler is an operating system utility, this is inherently not the case – in Huang, it is possible for one thread of one VPS to monopolize the CPU time, or for one thread of one VPS to “steal” CPU time for another VPS. Since in the recited combination, this is not true, this provides yet another distinction over Huang, or Huang in combination with the other references.

Additionally, all the independent claims recite an application interface that includes system calls, shared memory interfaces, and I/O driver controls. This feature is also not disclosed in Huang, providing yet another distinction over Huang, or any combination of Huang with other references.

Amended Claim 5

Claim 5 has been amended, to now recite the last clause of original claim 13. This aspect was discussed during the interview in some detail. Also, this feature was cancelled from claim 13, to avoid duplication. As discussed during the interview, this feature allows a great deal of flexibility for the host and service provider, when it comes to meeting (or occasionally not meeting) the terms of the service level agreement. None of the references, singly or in combination, disclose this aspect. Therefore, claim 5 is allowable for this additional reason as well.

Amended Claim 6

Claim 6 has been amended to recite one of the features previously recited in original claim 11. The feature was cancelled from original claim 11, to avoid duplication. As also discussed during the interview, this feature was not disclosed by Huang, or Huang in combination with other references. Applicants therefore respectfully submit that claim 6 is allowable for this additional reason as well.

Conclusion

All of the stated grounds of objection and rejection have been properly traversed, accommodated, or rendered moot. Applicants therefore respectfully request that the Examiner reconsider all presently outstanding objections and rejections and that they be withdrawn. Applicants believe that a full and complete reply has been made to the outstanding Office Action and, as such, the present application is in condition for allowance. If the Examiner believes, for any reason, that personal communication will expedite prosecution of this application, the Examiner is invited to telephone the undersigned at the number provided.

- 25 -

Belousov *et al.*
Appl. No. 10/703,594

Prompt and favorable consideration of this Amendment and Reply is respectfully
requested.

Respectfully submitted,

BARDMESSER LAW GROUP

/GB/

George S. Bardmesser
Attorney for Applicants
Registration No. 44,020

Date: June 6, 2008

910 17th Street, N.W., Suite 800
Washington, D.C. 20006
(202) 293-1191

Electronic Acknowledgement Receipt

EFS ID:	3417082
Application Number:	10703594
International Application Number:	
Confirmation Number:	2218
Title of Invention:	Virtual private server with isolation of system components
First Named Inventor/Applicant Name:	Serguei M. Beloussov
Customer Number:	54089
Filer:	George Simon Bardmesser
Filer Authorized By:	
Attorney Docket Number:	2230.0010003/MBR/GSB
Receipt Date:	06-JUN-2008
Filing Date:	10-NOV-2003
Time Stamp:	15:24:02
Application Type:	Utility under 35 USC 111(a)

Payment information:

Submitted with Payment	no
------------------------	----

File Listing:

Document Number	Document Description	File Name	File Size(Bytes) /Message Digest	Multi Part /.zip	Pages (if appl.)
1		AMENDMENTINRESPONS ETOFIRSTOFFICEACTION. pdf	156285 <small>65c88c44bf9cb6e14c067cdc05dd30e9 57fefe34</small>	yes	25

Multipart Description/PDF files in .zip description			
Document Description		Start	End
Amendment - After Non-Final Rejection		1	2
Abstract		3	3
Claims		4	20
Applicant Arguments/Remarks Made in an Amendment		21	25

Warnings:**Information:**

Total Files Size (in bytes):	156285
-------------------------------------	--------

This Acknowledgement Receipt evidences receipt on the noted date by the USPTO of the indicated documents, characterized by the applicant, and including page counts, where applicable. It serves as evidence of receipt similar to a Post Card, as described in MPEP 503.

New Applications Under 35 U.S.C. 111

If a new application is being filed and the application includes the necessary components for a filing date (see 37 CFR 1.53(b)-(d) and MPEP 506), a Filing Receipt (37 CFR 1.54) will be issued in due course and the date shown on this Acknowledgement Receipt will establish the filing date of the application.

National Stage of an International Application under 35 U.S.C. 371

If a timely submission to enter the national stage of an international application is compliant with the conditions of 35 U.S.C. 371 and other applicable requirements a Form PCT/DO/EO/903 indicating acceptance of the application as a national stage submission under 35 U.S.C. 371 will be issued in addition to the Filing Receipt, in due course.

New International Application Filed with the USPTO as a Receiving Office

If a new international application is being filed and the international application includes the necessary components for an international filing date (see PCT Article 11 and MPEP 1810), a Notification of the International Application Number and of the International Filing Date (Form PCT/RO/105) will be issued in due course, subject to prescriptions concerning national security, and the date shown on this Acknowledgement Receipt will establish the international filing date of the application.

PTO/SB/06 (07-06)

Approved for use through 1/31/2007. OMB 0651-0032
U.S. Patent and Trademark Office; U.S. DEPARTMENT OF COMMERCE

Under the Paperwork Reduction Act of 1995, no persons are required to respond to a collection of information unless it displays a valid OMB control number.

PATENT APPLICATION FEE DETERMINATION RECORD Substitute for Form PTO-875					Application or Docket Number 10/703,594		Filing Date 11/10/2003		<input type="checkbox"/> To be Mailed		
APPLICATION AS FILED – PART I											
(Column 1)			(Column 2)			SMALL ENTITY <input checked="" type="checkbox"/> OR		OTHER THAN SMALL ENTITY			
FOR	NUMBER FILED	NUMBER EXTRA	RATE (\$)	FEE (\$)	OR	RATE (\$)	FEE (\$)				
<input type="checkbox"/> BASIC FEE (37 CFR 1.16(a), (b), or (c))	N/A	N/A	N/A			N/A					
<input type="checkbox"/> SEARCH FEE (37 CFR 1.16(k), (l), or (m))	N/A	N/A	N/A			N/A					
<input type="checkbox"/> EXAMINATION FEE (37 CFR 1.16(o), (p), or (q))	N/A	N/A	N/A			N/A					
TOTAL CLAIMS (37 CFR 1.16(i))	minus 20 =	*	X \$ =		OR	X \$ =					
INDEPENDENT CLAIMS (37 CFR 1.16(h))	minus 3 =	*	X \$ =			X \$ =					
<input type="checkbox"/> APPLICATION SIZE FEE (37 CFR 1.16(s))	If the specification and drawings exceed 100 sheets of paper, the application size fee due is \$250 (\$125 for small entity) for each additional 50 sheets or fraction thereof. See 35 U.S.C. 41(a)(1)(G) and 37 CFR 1.16(s).										
<input type="checkbox"/> MULTIPLE DEPENDENT CLAIM PRESENT (37 CFR 1.16(j))											
* If the difference in column 1 is less than zero, enter "0" in column 2.			TOTAL			TOTAL					
APPLICATION AS AMENDED – PART II											
(Column 1)			(Column 2)			SMALL ENTITY OR		OTHER THAN SMALL ENTITY			
AMENDMENT	06/06/2008	CLAIMS REMAINING AFTER AMENDMENT		HIGHEST NUMBER PREVIOUSLY PAID FOR	PRESENT EXTRA	RATE (\$)	ADDITIONAL FEE (\$)	OR	RATE (\$)	ADDITIONAL FEE (\$)	
	Total (37 CFR 1.16(i))	* 80	Minus	** 80	=	X \$ =		OR	X \$ =		
	Independent (37 CFR 1.16(h))	* 12	Minus	*** 12	=	X \$ =		OR	X \$ =		
	<input type="checkbox"/> Application Size Fee (37 CFR 1.16(s))								OR		
	<input type="checkbox"/> FIRST PRESENTATION OF MULTIPLE DEPENDENT CLAIM (37 CFR 1.16(j))								OR		
						TOTAL ADD'L FEE		OR	TOTAL ADD'L FEE		
AMENDMENT		CLAIMS REMAINING AFTER AMENDMENT		HIGHEST NUMBER PREVIOUSLY PAID FOR	PRESENT EXTRA	RATE (\$)	ADDITIONAL FEE (\$)	OR	RATE (\$)	ADDITIONAL FEE (\$)	
	Total (37 CFR 1.16(i))	*	Minus	**	=	X \$ =		OR	X \$ =		
	Independent (37 CFR 1.16(h))	*	Minus	***	=	X \$ =		OR	X \$ =		
	<input type="checkbox"/> Application Size Fee (37 CFR 1.16(s))								OR		
	<input type="checkbox"/> FIRST PRESENTATION OF MULTIPLE DEPENDENT CLAIM (37 CFR 1.16(j))								OR		
						TOTAL ADD'L FEE		OR	TOTAL ADD'L FEE		
<p>* If the entry in column 1 is less than the entry in column 2, write "0" in column 3.</p> <p>** If the "Highest Number Previously Paid For" IN THIS SPACE is less than 20, enter "20".</p> <p>*** If the "Highest Number Previously Paid For" IN THIS SPACE is less than 3, enter "3".</p> <p>The "Highest Number Previously Paid For" (Total or Independent) is the highest number found in the appropriate box in column 1.</p>											

Legal Instrument Examiner:
/ELMIRA HALL/

This collection of information is required by 37 CFR 1.16. The information is required to obtain or retain a benefit by the public which is to file (and by the USPTO to process) an application. Confidentiality is governed by 35 U.S.C. 122 and 37 CFR 1.14. This collection is estimated to take 12 minutes to complete, including gathering, preparing, and submitting the completed application form to the USPTO. Time will vary depending upon the individual case. Any comments on the amount of time you require to complete this form and/or suggestions for reducing this burden, should be sent to the Chief Information Officer, U.S. Patent and Trademark Office, U.S. Department of Commerce, P.O. Box 1450, Alexandria, VA 22313-1450. DO NOT SEND FEES OR COMPLETED FORMS TO THIS ADDRESS. **SEND TO: Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450.**

If you need assistance in completing the form, call 1-800-PTO-9199 and select option 2.

Notice of Allowability	Application No.	Applicant(s)	
	10/703,594	BELOUSSOV ET AL.	
	Examiner	Art Unit	
	MELVIN H. POLLACK	2145	

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address--

All claims being allowable, PROSECUTION ON THE MERITS IS (OR REMAINS) CLOSED in this application. If not included herewith (or previously mailed), a Notice of Allowance (PTOL-85) or other appropriate communication will be mailed in due course. **THIS NOTICE OF ALLOWABILITY IS NOT A GRANT OF PATENT RIGHTS.** This application is subject to withdrawal from issue at the initiative of the Office or upon petition by the applicant. See 37 CFR 1.313 and MPEP 1308.

1. ☒ This communication is responsive to the amendment filed 06 June 2008.
2. ☐ The allowed claim(s) is/are 1-80.
3. ☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).
- a) ☐ All b) ☐ Some* c) ☐ None of the:
- ☐ Certified copies of the priority documents have been received.
 - ☐ Certified copies of the priority documents have been received in Application No. _____.
 - ☐ Copies of the certified copies of the priority documents have been received in this national stage application from the International Bureau (PCT Rule 17.2(a)).

* Certified copies not received: _____.

Applicant has THREE MONTHS FROM THE "MAILING DATE" of this communication to file a reply complying with the requirements noted below. Failure to timely comply will result in ABANDONMENT of this application.

THIS THREE-MONTH PERIOD IS NOT EXTENDABLE.

4. ☐ A SUBSTITUTE OATH OR DECLARATION must be submitted. Note the attached EXAMINER'S AMENDMENT or NOTICE OF INFORMAL PATENT APPLICATION (PTO-152) which gives reason(s) why the oath or declaration is deficient.
5. ☐ CORRECTED DRAWINGS (as "replacement sheets") must be submitted.
- (a) ☐ including changes required by the Notice of Draftsperson's Patent Drawing Review (PTO-948) attached
- ☐ hereto or 2) ☐ to Paper No./Mail Date _____.
- (b) ☐ including changes required by the attached Examiner's Amendment / Comment or in the Office action of Paper No./Mail Date _____.
- Identifying indicia such as the application number (see 37 CFR 1.84(c)) should be written on the drawings in the front (not the back) of each sheet. Replacement sheet(s) should be labeled as such in the header according to 37 CFR 1.121(d).**
6. ☐ DEPOSIT OF and/or INFORMATION about the deposit of BIOLOGICAL MATERIAL must be submitted. Note the attached Examiner's comment regarding REQUIREMENT FOR THE DEPOSIT OF BIOLOGICAL MATERIAL.

Attachment(s)

- | | |
|--|--|
| 1. <input checked="" type="checkbox"/> Notice of References Cited (PTO-892) | 5. <input type="checkbox"/> Notice of Informal Patent Application |
| 2. <input type="checkbox"/> Notice of Draftsperson's Patent Drawing Review (PTO-948) | 6. <input type="checkbox"/> Interview Summary (PTO-413),
Paper No./Mail Date _____. |
| 3. <input type="checkbox"/> Information Disclosure Statements (PTO/SB/08),
Paper No./Mail Date _____ | 7. <input type="checkbox"/> Examiner's Amendment/Comment |
| 4. <input type="checkbox"/> Examiner's Comment Regarding Requirement for Deposit
of Biological Material | 8. <input checked="" type="checkbox"/> Examiner's Statement of Reasons for Allowance |
| | 9. <input checked="" type="checkbox"/> Other <u>see attached office action</u> . |

/M. H. P./
Examiner, Art Unit 2145

Application/Control Number: 10/703,594
Art Unit: 2145

Page 2

DETAILED ACTION

Allowable Subject Matter

1. Claims 1-80 are allowed.
2. The following is an examiner's statement of reasons for allowance: after the updated search and consideration, examiner has decided that the amended claims, in light of the remarks, are allowable for reasons discussed in the interview summary.
3. The claims are drawn to a method and system of placing multiple and separate VPS on a single server sharing a single kernel such that the usage of one VPS cannot adversely impact the parallel usage of a different VPS. Applicant has successfully moved away from the field of virtual machines, particularly in the use of a scheduler and a single instance of an OS. Of the known art involving Virtual Private Servers (VPS), the combination of limitations is novel and non-obvious.

Any comments considered necessary by applicant must be submitted no later than the payment of the issue fee and, to avoid processing delays, should preferably accompany the issue fee. Such submissions should be clearly labeled "Comments on Statement of Reasons for Allowance."

Conclusion

4. The prior art made of record and not relied upon is considered pertinent to applicant's disclosure. They regard further background teachings on VPSs, OS kernels, and schedulers.

Any inquiry concerning this communication or earlier communications from the examiner should be directed to MELVIN H. POLLACK whose telephone number is (571)272-3887. The examiner can normally be reached on 8:00-4:30 M-F.

Application/Control Number: 10/703,594
Art Unit: 2145

Page 3

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Jason Cardone can be reached on (571) 272-3933. The fax phone number for the organization where this application or proceeding is assigned is 571-273-8300.

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see <http://pair-direct.uspto.gov>. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free). If you would like assistance from a USPTO Customer Service Representative or access to the automated information system, call 800-786-9199 (IN USA OR CANADA) or 571-272-1000.

/M. H. P./
Examiner, Art Unit 2145
25 August 2008

/Jason D Cardone/
Supervisory Patent Examiner, Art Unit 2145

INVENTOR(S)/APPLICANT(S)				
LAST NAME	FIRST NAME	MIDDLE INITIAL	RESIDENCE (CITY AND EITHER STATE OR FOREIGN COUNTRY)	
Tormasov	Alexander		Moscow, Russia	
Lunev	Dennis		Moscow, Russia	
Beloussov	Serguei		Singapore, Singapore	
Protassov	Stanislav		Moscow, Russia	
Pudgorodsky	Yuri		Moscow, Russia	
TITLE OF THE INVENTION (280 characters max)				
USE OF VIRTUAL COMPUTING ENVIRONMENTS TO PROVIDE FULL INDEPENDENT OPERATING SYSTEM SERVICES ON A SINGLE HARDWARE NODE				
CORRESPONDENCE ADDRESS				
Alan R. Thiele JENKENS & GILCHRIST 1445 Ross Avenue, Suite 3200 Dallas, Texas 75202-2799				
STATE	Texas	ZIP CODE	75202-2799	
COUNTRY				US
ENCLOSED APPLICATION PARTS (check all that apply)				
<input checked="" type="checkbox"/> Specification	Number of Pages	10	<input checked="" type="checkbox"/> Small Entity Statement	
<input checked="" type="checkbox"/> Drawing(s)	Number of Sheets	3	<input checked="" type="checkbox"/> Other (Specify) Express Mail Certificate	
METHOD OF PAYMENT OF FILING FEES FOR THIS PROVISIONAL APPLICATION FOR PATENT (check one)				
<input checked="" type="checkbox"/> A check or money order is enclosed to cover the provisional filing fees	PROVISIONAL FILING FEE AMOUNT (\$)		\$75.00	
<input type="checkbox"/> The Commissioner is hereby authorized to charge filing fees and credit Deposit Account Number:	10-0447			

This invention was made by an agency of the United States Government or under contract with an agency of the United States Government

☒ No.

☐ Yes, the name of the U.S. Government/agency and the Government contract number are:

Respectfully submitted,

SIGNATURE [Signature] Date 2/16/00

TYPED or PRINTED NAME Alan R. Thiele REGISTRATION NO. 30,694
(if appropriate)

☐ Additional inventors are being named on separately numbered sheets attached hereto.

USE ONLY FOR FILING A PROVISIONAL APPLICATION FOR PATENT

Provisional Patent Application
Applicant: Alexander Tormasov, et al.
Attorney Docket No.: 44151.00006

**PROVISIONAL PATENT APPLICATION
OF
ALEXANDER TORMASOV,
DENIS LUNEV,
YURI PUDGORODSKY,
SERGEI BELOUSSOV,
AND
STANISLAV PROTASSOV**

ENTITLED:

**USE OF VIRTUAL COMPUTING ENVIRONMENTS TO PROVIDE
FULL INDEPENDENT OPERATING SYSTEM SERVICES ON A SINGLE
HARDWARE NODE**

CERTIFICATE OF MAILING 37 C.F.R. § 1.10

I hereby certify that this correspondence is being deposited with the United States Postal Service with sufficient postage as Express Mail, Express Mail No. EL525151963US, addressed to: BOX PROVISIONAL PATENT APPLICATION, Commissioner of Patents, Washington, D.C. 20231, on February 16, 2001.

Melanie Reese Capps

Printed Name of Person

Melanie Reese Capps
Signature

BACKGROUND OF THE INVENTION

Technical Field

This invention relates to providing full independent computer system services
5 across a network of remote computer connections.

Description of the Prior Art

The problem of providing computer services across remote computer connections
has existed during the last 30-40 years beginning with the early stages of computer
technologies. In the very beginning, during the mainframe computer age, this problem
10 was solved by renting computer terminals which belong to a mainframe and connected to
it via modem or dedicated lines and provide some mainframe access services. Later, with
the beginning of the age of personal computers and with the widespread acceptance of the
client-server model -- the problem of access to large information sources, at first look,
seems to have been solved -- everyone could have his own computer and then rent an
15 Internet connection to obtain access to information sources on other computers.

Today, with wide growth of Internet access, another problem has arisen -- the
problem of creation of the information sources. Usually users want to put out their own
information sources in the form of websites somewhere and provide other computer users
with access to these websites. However, it is not possible to install a web server on most
20 home connections to a personal computer, simply because usually the connection to the
network from a home computer is simply too weak. There is a big industry called a
"hosting service" which provides computer users with an ability to utilize installed web
services.

Usually when one wants to provide Internet users with some information, which information could be of interest for a wide range of Internet users (usually in web server form), one must both put the information somewhere, and one must also provide a reliable network connection to the information.

5 The problem of providing access by ordinary users to large capacity computers appeared practically at the moment of the beginning of their industrial production. In the epoch of the mainframes, when the access by all users directly to computer equipment was difficult, this access problem was solved by providing users with remote terminals. These remote terminals were used to obtain certain services from mainframe computers.

10 The advantage of using remote terminals with a mainframe computer was that the user had little trouble accessing both the mainframe computer hardware and to some extent accessing the software resident on the mainframe computer. This is because the administration of the operation of a mainframe computer has always dealt with the installation and updating of software.

15 Further on, in the epoch of personal computers, each user could gain access to computing power directly from his workplace or home. With the advent of access to the Internet, this situation could probably cover the needs of most all users for both large amounts of information and robust operating systems.

20 The client-server model of networking computers provides an access system in which a personal computer is designated as the client computer and another computer or a set of computers is designated as the server computer -- access to which is carried out in a remote way covering the majority of needs of the common computer users.

But even the client-server model has some very fundamental drawbacks. Specifically, the high price of servicing of many client workplace computers, which includes creating a network infrastructure and the installation and the upgrading of software and hardware to obtain the bandwidth of network access for client computers, is a significant drawback. Additionally, the rapid growth of information available worldwide on the Internet has produced more users who in turn place even more information on the Internet. The required service to client computers should be provided by a server computer (most often it is web or www server) which, by itself, is powerful enough and is provided with an access channel to the Internet with corresponding power. Usually, personal computers of users have enough performance capability to interact with most of the web servers, but the typical channel of access to the network is most often one or two orders less productive than is needed. Besides, most often the personal computers in most homes cannot provide a sufficient level of service reliability and good level of security, etc. The same problems, besides the Internet services, appear for common users when very complex software packages are used. Users spend a lot of effort setting up and administering these complex software packages. To solve the given problems with web service, the technologies of using remote web hosting are usually applied wherein some company (usually ISP Internet Service Providers) provide the services of hosting of the web servers for the users. In that case the user is restricted to utilization of the standard preinstalled web server of the ISP. As a result the user is restricted in his possibilities.

Problems usually arise with the use of CGI scripts and more complex applications requiring, for example, a data base. Such computer tools can't be used for access to any

program of the user set on a remote server. The reason is that the user has become used to the absolute freedom in the adjustment of his local machine. But, the limitations that are imposed upon a user by the administration of the remote node are often unacceptable to a user.

5 The reasonable solution of such problem is by the use of emulators of computers. The operating system for IBM mainframe computers is named OS/390 and has been used for many years. Each user is given his own fully-functional virtual computer with emulated hardware. The implementation of the disclosed approach is connected with high costs, as every operating system installed in the corresponding virtual computer does
10 not know anything about the existence of the neighboring analogous computers and shares practically no resources with neighboring computers. Experience has shown that the price associated with virtual computers is very great.

Another analogous solution for non-mainframe computers utilizes software emulators of the VMware type. The software programs of the VMware type exist for
15 different types of operating systems and wholly emulate some typical computer inside one process of the main computer operating system.

The main problem is that not many of such computer emulators on a server having a typical configuration can be used. Usually the limitations on the use of computer emulators are connected with the fact that the size of the emulated memory is
20 close to the size of the memory used by the process or in which the computer emulator works. That is, the number of computer emulators that can be used on one server simultaneously is confined to a numeric range from 2-3 to 10-15. All the above mentioned solutions could be classified as multikernel implementations of virtual

computers when on one physical computer there are simultaneously several kernels of operating systems that are unaware of each other.

So, when it is necessary for many users to deal with a hosting computer, there is a necessity to provide each user with a complete set of services that the user can expect from the computer (the provision of virtual environments emulating with maximal completeness of the whole computer with the operating system installed in it). For the purpose of effectiveness of the use of equipment, it is desirable that the number of such computers in a virtual environment installed in one computer should be two or three orders more than in the above mentioned cases.

10 **BRIEF SUMMARY OF THE INVENTION**

The present invention describes a method of efficient utilization of a single hardware system with a single Operating System kernel. The end user is provided with a virtual computing environment that is functionally equivalent to a computer with a full-featured Operating System. There is no emulation of hardware or dedicated physical memory or any other hardware resources as in the case of solution of VMware kind.

The method of the present invention is implemented by the separation of user processes on the level of namespace and on the basis of restrictions inside the Operating System kernel. Virtual computing environment processes are never visible to other virtual computing environments running on the same computer. A virtual computing environment root file system is also never visible to other virtual computing environments running on the same computer. The root file system of a virtual computing environment is working in the read-write mode and allows the root user of every virtual computing environment to perform file modifications and Operating System configuration.

BRIEF DESCRIPTION OF THE SEVERAL VIEWS OF THE DRAWING

A better understanding of the present invention may be had by reference to the drawing figures, wherein:

5 FIG. 1 shows a network of end users with access to virtual computing environments encapsulated in a computer with a full feature operating system in accordance with the present invention;

 FIG. 2 shows a utilization of resources of hardware (memory and file system) by different virtual computing environments; and

10 FIG. 3 shows a utilization of resources of hardware (memory and file system) in another full hardware emulation solution.

DETAILED DESCRIPTION OF THE INVENTION

 The disclosed invention presents the method of efficient utilization of a single
15 hardware system with a single Operating System kernel.

 The goal of the present invention is to provide the possibility of work on a system that is the functional equivalent of a computer with a full-featured Operating system.

 As a result the possibility appears to the user as if he has obtained full network root access to a common computer with fully-featured Operating System installed on it.

20 Specifically, the end user is provided with a virtual computing environment that is functionally equivalent to a computer with full-featured Operating System.

 From the point of view of the end user, each virtual computing environment is the actual remote computer with the network address in which the end user can perform all actions allowed for the ordinary computer: the work in command shells, compilation and
25 installation of programs, configuration of network services, offices and other

applications. Several different users can work with the same hardware node without noticing each other in the same way as if they worked on different computers (Fig. 1).

Each virtual computing environment includes a complete set of processes and files of an operating system that can be modified by the end user.

5 In addition the end user may stop and start the virtual computing environment in the same way that is done with a common operating system. Nevertheless all virtual computing environments share one and the same kernel of the operating system. All the processes inside the virtual computing environment are the common processes of the operating system and all the resources inherent to each virtual computing environment
10 are shared in the same way as typically happens inside an ordinary single kernel operating system.

Fig. 2 shows the process of the coexistence of the two virtual computing environments on one hardware computer. Each virtual computing environment has its own unique file system and each virtual environment can also see the common file
15 system. All the processes of all virtual computing environments work from inside the same physical memory. If two processes in different virtual computing environments were started for execution from one file (for example from the shared file system) they would be completely isolated from each other.

In such a way the high effectiveness of implementation of multiple virtual
20 computing environments inside one operating system is reached.

There is no emulation of hardware or dedicated physical memory or another hardware resource.

The disclosed invention differs from the other solutions (Fig. 3) that provide a complete emulation of computer hardware and give the user the full scope virtual
25 computer with the doubled expenses. This happens because a minimum of 2 actual

kernels are performed in the computer, one inside the other - the kernel of the main operating system and inside the process, the kernel of the emulated operating system.

The implementation of the kernel of the Operating System with the properties necessary for this invention should carry out the separation of the users not on the level of
5 hardware but on the level of the namespace and on the basis of limitations implemented inside the kernel of the Operating System.

Virtual computing environment processes are never visible to other virtual computing environments running on the same computer. The virtual computing environment root file system is also never visible to other virtual computing
10 environments running on the same computer. The root file system of the virtual computing environment is working in the read-write mode and allows a root user of every virtual computing environment to make file modifications and configure the operating system.

The changes done in the file system in one virtual computing environment do not
15 influence the file systems in the other virtual computing environment.

CLAIMS

We claim:

1 1. A method for efficient utilization of a single hardware system with a
2 single operating system kernel including a virtual computing environment functionally
3 equivalent to computer having a full-featured operating system is provided to an end user
4 without emulation of hardware or dedicated physical memory or hardware resource, and
5 where such method is realized by separation of user processes on the level of namespace
6 and on the base of restrictions implemented inside said operating system kernel.

1 2. The method as defined in Claim 1 wherein each virtual computing
2 environment is not visible to other virtual computing environments on the system.

1 3. The method as defined in Claim 1 wherein each virtual computing
2 environment has a completely independent root file system.

ABSTRACT OF THE DISCLOSURE

Method of efficient utilization of a single hardware system with single Operating System kernel wherein a Virtual Environment, functionally equivalent to full-featured Operating System box, is provided to an end user without emulation of hardware or
5 dedicated physical memory or another hardware resource. Such method is realized by separation of user processes on the level of namespace and on the basis of restrictions implemented inside the Operating system kernel. Each Virtual Environment is not visible to another Virtual Environment within the system and has a completely independent root file system.

The work of end users with virtual environments
encapsulated in the same hardware box

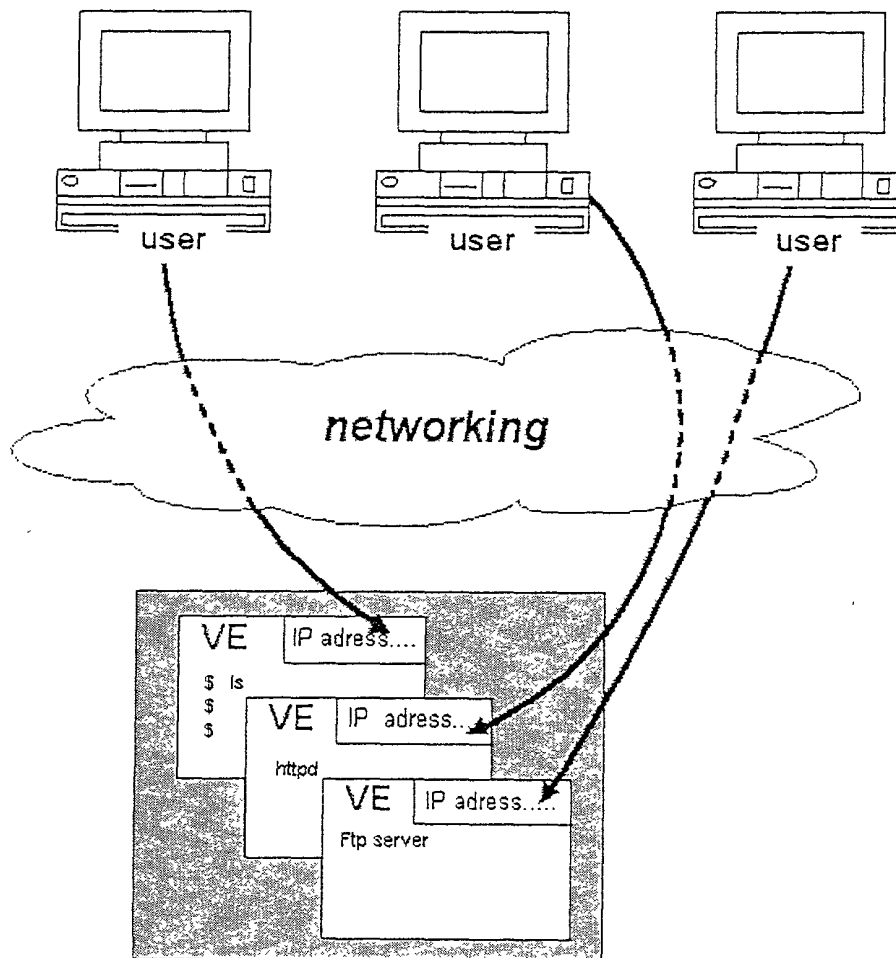


FIG 1

Utilization of resources of hardware (memory and file system) by different Virtual Environment's

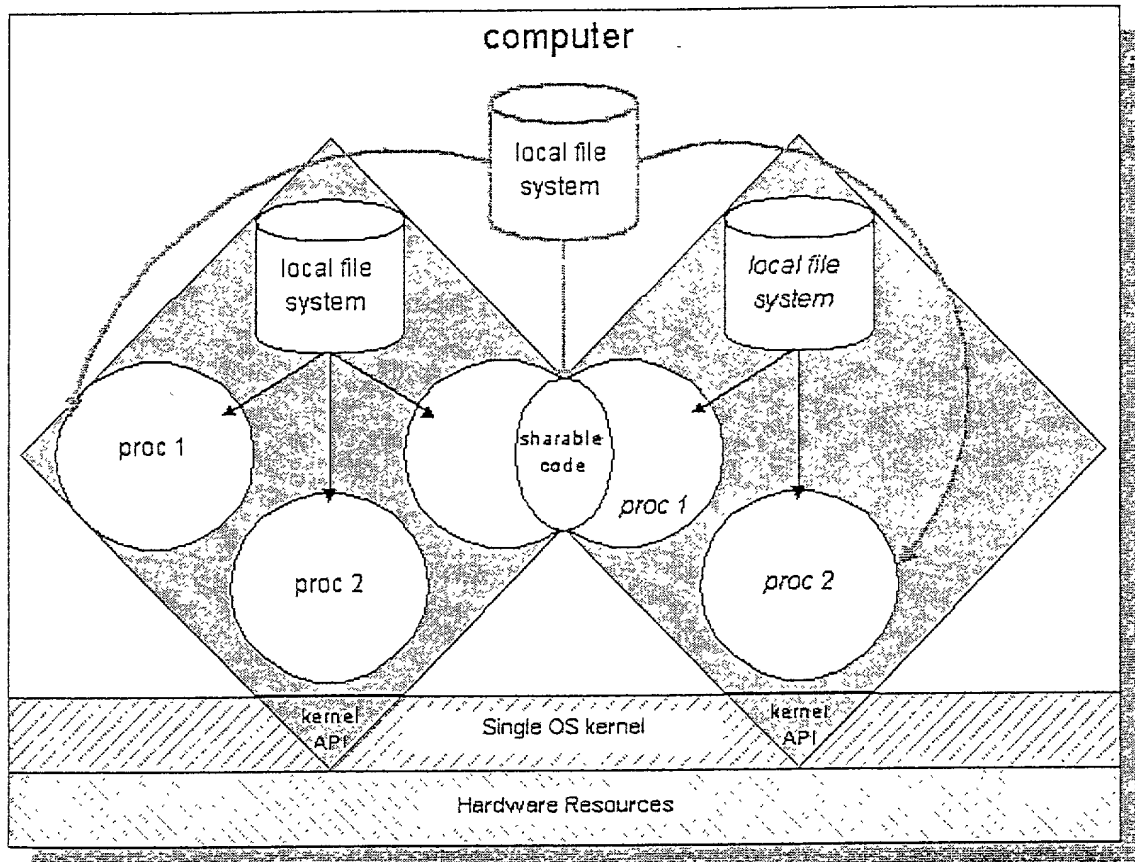


FIG 2

Utilization of resources of hardware (memory and file system) in another full hardware emulation solutions

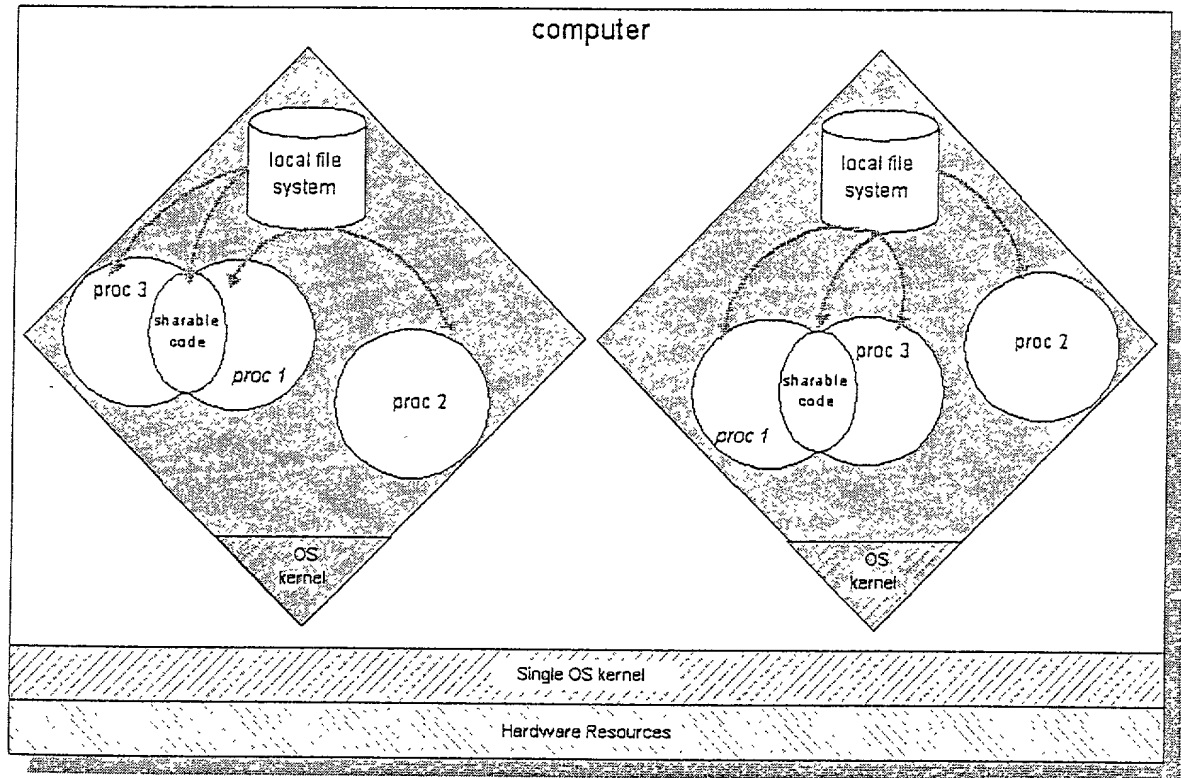
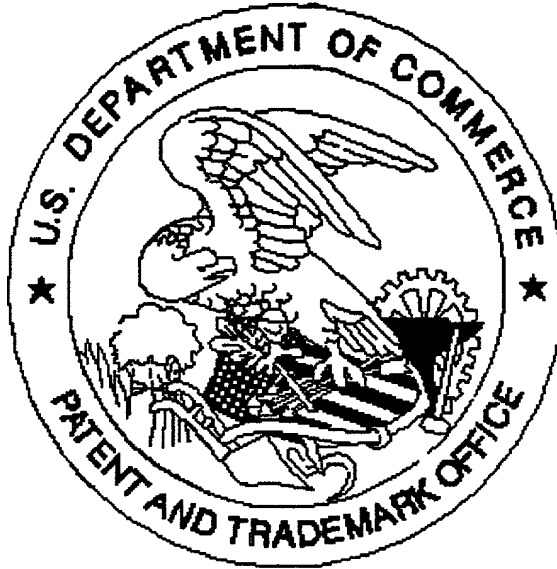


FIG 3

United States Patent & Trademark Office
Office of Initial Patent Examination -- Scanning Division



Application deficiencies found during scanning:

☐ Page(s) _____ of small entity statement were not present
for scanning. (Document title)

☐ Page(s) _____ of _____ were not present
for scanning. (Document title)

☐ *Scanned copy is best available.*

EXHIBIT I



UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
 United States Patent and Trademark Office
 Address: COMMISSIONER FOR PATENTS
 P.O. Box 1450
 Alexandria, Virginia 22313-1450
 www.uspto.gov

APPLICATION NUMBER	FILING or 371(c) DATE	GRP ART UNIT	FIL FEE REC'D	ATTY. DOCKET NO	TOT CLAIMS	IND CLAIMS
62/580,611	11/02/2017		260	038179-00027		

38485
 ARENT FOX LLP
 1675 BROADWAY
 NEW YORK, NY 10019

CONFIRMATION NO. 1016
FILING RECEIPT



CC000000096778170

Date Mailed: 01/19/2018

Receipt is acknowledged of this provisional patent application. It will not be examined for patentability and will become abandoned not later than twelve months after its filing date. Any correspondence concerning the application must include the following identification information: the U.S. APPLICATION NUMBER, FILING DATE, NAME OF APPLICANT, and TITLE OF INVENTION. Fees transmitted by check or draft are subject to collection. Please verify the accuracy of the data presented on this receipt. **If an error is noted on this Filing Receipt, please submit a written request for a Filing Receipt Correction. Please provide a copy of this Filing Receipt with the changes noted thereon. If you received a "Notice to File Missing Parts" for this application, please submit any corrections to this Filing Receipt with your reply to the Notice. When the USPTO processes the reply to the Notice, the USPTO will generate another Filing Receipt incorporating the requested corrections**

Inventor(s)

Stanislav Kinsburskiy, Moscow, RUSSIAN FEDERATION;
 Alexey Kobets, Seattle, WA;
 Eugene Kolomeetz, Moscow, RUSSIAN FEDERATION;

Applicant(s)

Virtuozzo International GmbH, Schaffhausen, SWITZERLAND;

Power of Attorney: None

Permission to Access Application via Priority Document Exchange: Yes

Permission to Access Search Results: Yes

Applicant may provide or rescind an authorization for access using Form PTO/SB/39 or Form PTO/SB/69 as appropriate.

If Required, Foreign Filing License Granted: 01/18/2018

The country code and number of your priority application, to be used for filing abroad under the Paris Convention, is **US 62/580,611**

Projected Publication Date: None, application is not eligible for pre-grant publication

Non-Publication Request: No

Early Publication Request: No

Title

SYSTEM AND METHOD FOR USERSPACE LIVE PATCHING

Statement under 37 CFR 1.55 or 1.78 for AIA (First Inventor to File) Transition Applications: No

PROTECTING YOUR INVENTION OUTSIDE THE UNITED STATES

Since the rights granted by a U.S. patent extend only throughout the territory of the United States and have no effect in a foreign country, an inventor who wishes patent protection in another country must apply for a patent in a specific country or in regional patent offices. Applicants may wish to consider the filing of an international application under the Patent Cooperation Treaty (PCT). An international (PCT) application generally has the same effect as a regular national patent application in each PCT-member country. The PCT process **simplifies** the filing of patent applications on the same invention in member countries, but **does not result** in a grant of "an international patent" and does not eliminate the need of applicants to file additional documents and fees in countries where patent protection is desired.

Almost every country has its own patent law, and a person desiring a patent in a particular country must make an application for patent in that country in accordance with its particular laws. Since the laws of many countries differ in various respects from the patent law of the United States, applicants are advised to seek guidance from specific foreign countries to ensure that patent rights are not lost prematurely.

Applicants also are advised that in the case of inventions made in the United States, the Director of the USPTO must issue a license before applicants can apply for a patent in a foreign country. The filing of a U.S. patent application serves as a request for a foreign filing license. The application's filing receipt contains further information and guidance as to the status of applicant's license for foreign filing.

Applicants may wish to consult the USPTO booklet, "General Information Concerning Patents" (specifically, the section entitled "Treaties and Foreign Patents") for more information on timeframes and deadlines for filing foreign patent applications. The guide is available either by contacting the USPTO Contact Center at 800-786-9199, or it can be viewed on the USPTO website at <http://www.uspto.gov/web/offices/pac/doc/general/index.html>.

For information on preventing theft of your intellectual property (patents, trademarks and copyrights), you may wish to consult the U.S. Government website, <http://www.stopfakes.gov>. Part of a Department of Commerce initiative, this website includes self-help "toolkits" giving innovators guidance on how to protect intellectual property in specific countries such as China, Korea and Mexico. For questions regarding patent enforcement issues, applicants may call the U.S. Government hotline at 1-866-999-HALT (1-866-999-4258).

LICENSE FOR FOREIGN FILING UNDER

Title 35, United States Code, Section 184

Title 37, Code of Federal Regulations, 5.11 & 5.15

GRANTED

The applicant has been granted a license under 35 U.S.C. 184, if the phrase "IF REQUIRED, FOREIGN FILING LICENSE GRANTED" followed by a date appears on this form. Such licenses are issued in all applications where the conditions for issuance of a license have been met, regardless of whether or not a license may be required as set forth in 37 CFR 5.15. The scope and limitations of this license are set forth in 37 CFR 5.15(a) unless an earlier

license has been issued under 37 CFR 5.15(b). The license is subject to revocation upon written notification. The date indicated is the effective date of the license, unless an earlier license of similar scope has been granted under 37 CFR 5.13 or 5.14.

This license is to be retained by the licensee and may be used at any time on or after the effective date thereof unless it is revoked. This license is automatically transferred to any related applications(s) filed under 37 CFR 1.53(d). This license is not retroactive.

The grant of a license does not in any way lessen the responsibility of a licensee for the security of the subject matter as imposed by any Government contract or the provisions of existing laws relating to espionage and the national security or the export of technical data. Licensees should apprise themselves of current regulations especially with respect to certain countries, of other agencies, particularly the Office of Defense Trade Controls, Department of State (with respect to Arms, Munitions and Implements of War (22 CFR 121-128)); the Bureau of Industry and Security, Department of Commerce (15 CFR parts 730-774); the Office of Foreign Assets Control, Department of Treasury (31 CFR Parts 500+) and the Department of Energy.

NOT GRANTED

No license under 35 U.S.C. 184 has been granted at this time, if the phrase "IF REQUIRED, FOREIGN FILING LICENSE GRANTED" DOES NOT appear on this form. Applicant may still petition for a license under 37 CFR 5.12, if a license is desired before the expiration of 6 months from the filing date of the application. If 6 months has lapsed from the filing date of this application and the licensee has not received any indication of a secrecy order under 35 U.S.C. 181, the licensee may foreign file the application pursuant to 37 CFR 5.15(b).

SelectUSA

The United States represents the largest, most dynamic marketplace in the world and is an unparalleled location for business investment, innovation, and commercialization of new technologies. The U.S. offers tremendous resources and advantages for those who invest and manufacture goods here. Through SelectUSA, our nation works to promote and facilitate business investment. SelectUSA provides information assistance to the international investor community; serves as an ombudsman for existing and potential investors; advocates on behalf of U.S. cities, states, and regions competing for global investment; and counsels U.S. economic development organizations on investment attraction best practices. To learn more about why the United States is the best country in the world to develop technology, manufacture products, deliver services, and grow your business, visit <http://www.SelectUSA.gov> or call +1-202-482-6800.

Under the Paperwork Reduction Act of 1995, no persons are required to respond to a collection of information unless it contains a valid OMB control number.

Application Data Sheet 37 CFR 1.76		Attorney Docket Number	038179-00027
		Application Number	
Title of Invention	SYSTEM AND METHOD FOR USERSPACE LIVE PATCHING		
<p>The application data sheet is part of the provisional or nonprovisional application for which it is being submitted. The following form contains the bibliographic data arranged in a format specified by the United States Patent and Trademark Office as outlined in 37 CFR 1.76.</p> <p>This document may be completed electronically and submitted to the Office in electronic format using the Electronic Filing System (EFS) or the document may be printed and included in a paper filed application.</p>			

Secrecy Order 37 CFR 5.2:

☐ Portions or all of the application associated with this Application Data Sheet may fall under a Secrecy Order pursuant to 37 CFR 5.2 (Paper filers only. Applications that fall under Secrecy Order may not be filed electronically.)

Inventor Information:

Inventor 1					Remove
Legal Name					
Prefix	Given Name	Middle Name	Family Name	Suffix	
	Stanislav		Kinsburkiy		
Residence Information (Select One) <input type="radio"/> US Residency <input checked="" type="radio"/> Non US Residency <input type="radio"/> Active US Military Service					
City	Moscow	Country of Residenceⁱ	RU		
Mailing Address of Inventor:					
Address 1	c/o Virtuozzo International GmbH				
Address 2	Vordergasse 59				
City	Schaffhausen	State/Province			
Postal Code	8200	Countryⁱ	CH		
Inventor 2					Remove
Legal Name					
Prefix	Given Name	Middle Name	Family Name	Suffix	
	Alexey		Kobets		
Residence Information (Select One) <input checked="" type="radio"/> US Residency <input type="radio"/> Non US Residency <input type="radio"/> Active US Military Service					
City	Seattle	State/Province	WA	Country of Residenceⁱ	US
Mailing Address of Inventor:					
Address 1	c/o Virtuozzo International GmbH				
Address 2	Vordergasse 59				
City	Schaffhausen	State/Province			
Postal Code	8200	Countryⁱ	CH		
Inventor 3					Remove
Legal Name					
Prefix	Given Name	Middle Name	Family Name	Suffix	
	Eugene		Kolomeetz		

Application Data Sheet 37 CFR 1.76		Attorney Docket Number	038179-00027
		Application Number	
Title of Invention	SYSTEM AND METHOD FOR USERSPACE LIVE PATCHING		

Residence Information (Select One) <input type="radio"/> US Residency <input checked="" type="radio"/> Non US Residency <input type="radio"/> Active US Military Service			
City	Moscow	Country of Residence ⁱ	RU
Mailing Address of Inventor:			
Address 1	c/o Virtuozzo International GmbH		
Address 2	Vordergasse 59		
City	Schaffhausen	State/Province	
Postal Code	8200	Country ⁱ	CH
All Inventors Must Be Listed - Additional Inventor Information blocks may be generated within this form by selecting the Add button. Add			

Correspondence Information:

Enter either Customer Number or complete the Correspondence Information section below. For further information see 37 CFR 1.33(a).			
<input type="checkbox"/> An Address is being provided for the correspondence information of this application.			
Customer Number	38485		
Email Address	patentdocket@arentfox.com	Add Email	Remove Email

Application Information:

Title of the Invention	SYSTEM AND METHOD FOR USERSPACE LIVE PATCHING		
Attorney Docket Number	038179-00027	Small Entity Status Claimed	<input type="checkbox"/>
Application Type	Provisional		
Subject Matter	Utility		
Total Number of Drawing Sheets (if any)	5	Suggested Figure for Publication (if any)	

Filing By Reference:

Only complete this section when filing an application by reference under 35 U.S.C. 111(c) and 37 CFR 1.57(a). Do not complete this section if application papers including a specification and any drawings are being filed. Any domestic benefit or foreign priority information must be provided in the appropriate section(s) below (i.e., "Domestic Benefit/National Stage Information" and "Foreign Priority Information").

For the purposes of a filing date under 37 CFR 1.53(b), the description and any drawings of the present application are replaced by this reference to the previously filed application, subject to conditions and requirements of 37 CFR 1.57(a).

Application number of the previously filed application	Filing date (YYYY-MM-DD)	Intellectual Property Authority or Country

Application Data Sheet 37 CFR 1.76		Attorney Docket Number	038179-00027
		Application Number	
Title of Invention	SYSTEM AND METHOD FOR USERSPACE LIVE PATCHING		

Publication Information:

<input type="checkbox"/>	Request Early Publication (Fee required at time of Request 37 CFR 1.219)
<input type="checkbox"/>	Request Not to Publish. I hereby request that the attached application not be published under 35 U.S.C. 122(b) and certify that the invention disclosed in the attached application has not and will not be the subject of an application filed in another country, or under a multilateral international agreement, that requires publication at eighteen months after filing.


Representative Information:

Representative information should be provided for all practitioners having a power of attorney in the application. Providing this information in the Application Data Sheet does not constitute a power of attorney in the application (see 37 CFR 1.32). Either enter Customer Number or complete the Representative Name section below. If both sections are completed the customer Number will be used for the Representative Information during processing.			
Please Select One:			
<input checked="" type="radio"/>	Customer Number	<input type="radio"/>	US Patent Practitioner
<input type="radio"/>	Limited Recognition (37 CFR 11.9)		
Customer Number	38485		

Domestic Benefit/National Stage Information:

This section allows for the applicant to either claim benefit under 35 U.S.C. 119(e), 120, 121, 365(c), or 386(c) or indicate National Stage entry from a PCT application. Providing benefit claim information in the Application Data Sheet constitutes the specific reference required by 35 U.S.C. 119(e) or 120, and 37 CFR 1.78.

When referring to the current application, please leave the "Application Number" field blank.

Prior Application Status			
Application Number	Continuity Type	Prior Application Number	Filing or 371(c) Date (YYYY-MM-DD)
Additional Domestic Benefit/National Stage Data may be generated within this form by selecting the Add button.			

Foreign Priority Information:

This section allows for the applicant to claim priority to a foreign application. Providing this information in the application data sheet constitutes the claim for priority as required by 35 U.S.C. 119(b) and 37 CFR 1.55. When priority is claimed to a foreign application that is eligible for retrieval under the priority document exchange program (PDX) the information will be used by the Office to automatically attempt retrieval pursuant to 37 CFR 1.55(i)(1) and (2). Under the PDX program, applicant bears the ultimate responsibility for ensuring that a copy of the foreign application is received by the Office from the participating foreign intellectual property office, or a certified copy of the foreign priority application is filed, within the time period specified in 37 CFR 1.55(g)(1).

Application Data Sheet 37 CFR 1.76		Attorney Docket Number	038179-00027
		Application Number	
Title of Invention	SYSTEM AND METHOD FOR USERSPACE LIVE PATCHING		

Application Number	Country ⁱ	Filing Date (YYYY-MM-DD)	<div>Remove</div> Access Code ⁱ (if applicable)
Additional Foreign Priority Data may be generated within this form by selecting the Add button.			

Statement under 37 CFR 1.55 or 1.78 for AIA (First Inventor to File) Transition Applications

<input type="checkbox"/> This application (1) claims priority to or the benefit of an application filed before March 16, 2013 and (2) also contains, or contained at any time, a claim to a claimed invention that has an effective filing date on or after March 16, 2013. NOTE: By providing this statement under 37 CFR 1.55 or 1.78, this application, with a filing date on or after March 16, 2013, will be examined under the first inventor to file provisions of the AIA.

Application Data Sheet 37 CFR 1.76		Attorney Docket Number	038179-00027
		Application Number	
Title of Invention	SYSTEM AND METHOD FOR USERSPACE LIVE PATCHING		

Authorization or Opt-Out of Authorization to Permit Access:

When this Application Data Sheet is properly signed and filed with the application, applicant has provided written authority to permit a participating foreign intellectual property (IP) office access to the instant application-as-filed (see paragraph A in subsection 1 below) and the European Patent Office (EPO) access to any search results from the instant application (see paragraph B in subsection 1 below).

Should applicant choose not to provide an authorization identified in subsection 1 below, applicant **must opt-out** of the authorization by checking the corresponding box A or B or both in subsection 2 below.

NOTE: This section of the Application Data Sheet is **ONLY** reviewed and processed with the **INITIAL** filing of an application. After the initial filing of an application, an Application Data Sheet cannot be used to provide or rescind authorization for access by a foreign IP office(s). Instead, Form PTO/SB/39 or PTO/SB/69 must be used as appropriate.

1. Authorization to Permit Access by a Foreign Intellectual Property Office(s)

A. Priority Document Exchange (PDX) - Unless box A in subsection 2 (opt-out of authorization) is checked, the undersigned hereby **grants the USPTO authority** to provide the European Patent Office (EPO), the Japan Patent Office (JPO), the Korean Intellectual Property Office (KIPO), the State Intellectual Property Office of the People's Republic of China (SIPO), the World Intellectual Property Organization (WIPO), and any other foreign intellectual property office participating with the USPTO in a bilateral or multilateral priority document exchange agreement in which a foreign application claiming priority to the instant patent application is filed, access to: (1) the instant patent application-as-filed and its related bibliographic data, (2) any foreign or domestic application to which priority or benefit is claimed by the instant application and its related bibliographic data, and (3) the date of filing of this Authorization. See 37 CFR 1.14(h)(1).

B. Search Results from U.S. Application to EPO - Unless box B in subsection 2 (opt-out of authorization) is checked, the undersigned hereby **grants the USPTO authority** to provide the EPO access to the bibliographic data and search results from the instant patent application when a European patent application claiming priority to the instant patent application is filed. See 37 CFR 1.14(h)(2).

The applicant is reminded that the EPO's Rule 141(1) EPC (European Patent Convention) requires applicants to submit a copy of search results from the instant application without delay in a European patent application that claims priority to the instant application.

2. Opt-Out of Authorizations to Permit Access by a Foreign Intellectual Property Office(s)

A. Applicant **DOES NOT** authorize the USPTO to permit a participating foreign IP office access to the instant application-as-filed. ☐ If this box is checked, the USPTO will not be providing a participating foreign IP office with any documents and information identified in subsection 1A above.

B. Applicant **DOES NOT** authorize the USPTO to transmit to the EPO any search results from the instant patent application. ☐ If this box is checked, the USPTO will not be providing the EPO with search results from the instant application.

NOTE: Once the application has published or is otherwise publicly available, the USPTO may provide access to the application in accordance with 37 CFR 1.14.

Application Data Sheet 37 CFR 1.76		Attorney Docket Number	038179-00027
		Application Number	
Title of Invention	SYSTEM AND METHOD FOR USERSPACE LIVE PATCHING		

Applicant Information:

Providing assignment information in this section does not substitute for compliance with any requirement of part 3 of Title 37 of CFR to have an assignment recorded by the Office.

Applicant 1

If the applicant is the inventor (or the remaining joint inventor or inventors under 37 CFR 1.45), this section should not be completed. The information to be provided in this section is the name and address of the legal representative who is the applicant under 37 CFR 1.43; or the name and address of the assignee, person to whom the inventor is under an obligation to assign the invention, or person who otherwise shows sufficient proprietary interest in the matter who is the applicant under 37 CFR 1.46. If the applicant is an applicant under 37 CFR 1.46 (assignee, person to whom the inventor is obligated to assign, or person who otherwise shows sufficient proprietary interest) together with one or more joint inventors, then the joint inventor or inventors who are also the applicant should be identified in this section.

☒ Assignee ☐ Legal Representative under 35 U.S.C. 117 ☐ Joint Inventor

☐ Person to whom the inventor is obligated to assign. ☐ Person who shows sufficient proprietary interest

If applicant is the legal representative, indicate the authority to file the patent application, the inventor is:

Name of the Deceased or Legally Incapacitated Inventor:

If the Applicant is an Organization check here. ☒

Organization Name Virtuozzo International GmbH

Mailing Address Information For Applicant:

Address 1	Vordergasse 59		
Address 2			
City	Schaffhausen	State/Province	
Country	CH	Postal Code	8200
Phone Number		Fax Number	
Email Address			

Additional Applicant Data may be generated within this form by selecting the Add button.

Assignee Information including Non-Applicant Assignee Information:

Providing assignment information in this section does not substitute for compliance with any requirement of part 3 of Title 37 of CFR to have an assignment recorded by the Office.

Application Data Sheet 37 CFR 1.76		Attorney Docket Number	038179-00027
		Application Number	
Title of Invention	SYSTEM AND METHOD FOR USERSPACE LIVE PATCHING		

Assignee 1

Complete this section if assignee information, including non-applicant assignee information, is desired to be included on the patent application publication. An assignee-applicant identified in the "Applicant Information" section will appear on the patent application publication as an applicant. For an assignee-applicant, complete this section only if identification as an assignee is also desired on the patent application publication.

If the Assignee or Non-Applicant Assignee is an Organization check here. ☐

Prefix	Given Name	Middle Name	Family Name	Suffix

Mailing Address Information For Assignee including Non-Applicant Assignee:

Address 1			
Address 2			
City		State/Province	
Country ⁱ		Postal Code	
Phone Number		Fax Number	
Email Address			

Additional Assignee or Non-Applicant Assignee Data may be generated within this form by selecting the Add button.

Signature:

NOTE: This Application Data Sheet must be signed in accordance with 37 CFR 1.33(b). **However, if this Application Data Sheet is submitted with the INITIAL filing of the application and either box A or B is not checked in subsection 2 of the "Authorization or Opt-Out of Authorization to Permit Access" section, then this form must also be signed in accordance with 37 CFR 1.14(c).**

This Application Data Sheet **must** be signed by a patent practitioner if one or more of the applicants is a **juristic entity** (e.g., corporation or association). If the applicant is two or more joint inventors, this form must be signed by a patent practitioner, **all** joint inventors who are the applicant, or one or more joint inventor-applicants who have been given power of attorney (e.g., see USPTO Form PTO/AIA/81) on behalf of **all** joint inventor-applicants.

See 37 CFR 1.4(d) for the manner of making signatures and certifications.

Signature	/Michael Fainberg/		Date (YYYY-MM-DD)	2017-11-02
First Name	Michael	Last Name	Fainberg	Registration Number
50441				

Additional Signature may be generated within this form by selecting the Add button.

SYSTEM AND METHOD FOR USERSPACE LIVE PATCHING

Field of Technology

The invention is related to generating and applying binary patches to the code of running userspace applications.

Background

Why userspace patching is needed.

There are situations when it is very undesirable to stop and restart the application, while there is a necessity to update it. For example, when huge databases work with many clients, stopping and restarting will cause a downtime. Because time is needed for completing all tasks (queries), stopping processes, then starting new instance - i.e. new processes, putting needed information to the memory, etc. In this case, cache will be reset, so some time (e.g. the first few minutes) the application will work slower. But, sometimes critical updates, that should be applied immediately, appear. Usually, if such database works under load and has many clients, the usual solution is to wait. So a method, system and tool for creating and “painless” applying patches for userspace applications are needed.

Most of existing solutions for patching running processes (live patching) work only in OS kernel (Linux, Windows, etc.) and change only code in kernel space. For example, kpatch in Linux. Such technologies are being developed by RedHat, Suse, Oracle (the solution is called “Ksplice” is able to patch both kernel and user space), Cloud Linux (KernelCare) etc. But even solutions that are able to patch userspace have rather limited functionality and they are bounded to only one CPU architecture.

Existing solutions are implemented for only one CPU architecture each. The reason is that while creating a patch they rely on comparison of binary (or assembler) codes of 2 versions of the program (the old and the new ones). The task of comparing binary codes is complicated and has completely different solutions for different architectures. That is why existing solutions are usually bound to only one architecture. The patch in this case contains the difference between the binary code of the older version and the binary code the newer one. But, to allow

for creation of such patch, the both versions should be compiled in the same way. E.g. with precisely the same compiler options and often even with the same compiler version. In such cases, as much as possible instructions should occupy the same places in both versions of the program.

There are 2 main defects of conventional approach, which result from the need of having 2 similarly compiled versions of the program and comparing their binary code. The first one is the complexity of maintenance. If something in patch creation goes wrong, the maintenance of the program that creates patches becomes very complicated. Because, it is very hard to fix any mistakes in a program that creates patches by comparing binary code. The second one is that such approach usually works only for x86-64 platform. The reason is that the process of comparing binary code and creating the patch is complex, and is done by special program (e.g. comparator) the logic of which strongly depends on processor architecture.

Summary

Our method overcomes these constraints, works in user space and our method of patch creation is architecture independent (we can easily create patches for different processor architectures). (The module responsible for applying patches (further called "patcher") depends on the architecture, but it also could be easily changed for working on other architectures.) The patch in our solution is a shared object which is loaded to the process address space. We create a patch as a shared library (as such shared library could be mapped to any place in address space of a program being patched), which contains amended code (e.g. some functions). Amended functions and variables used in them that were static in the initial program now become an external (i.e. global) in a dynamically linked library. If some symbols (e.g. functions and variables) that were defined in the application (the program, which we patch) are also used in the patch, then they are defined in the patch as global external symbols. For example, a patch can be a shared library, containing fixed functions + references to static functions/variables in original binary (needed for fixed functions to work). To make these global variables point to a real process data, some instrumentation information has to be added to the

patch source code. This instrumentation information then has to be converted into binary hints used when patch is applied.

This shared object (library) is then dynamically linked to the process we want to patch. Special program module (in some aspects called "patcher") resolves such new dependencies. It has symbols resolving logic, similar to the one in dynamic linker.

Detailed Description

Components of the patching system:

Exemplary solution for live patching processes in userspace can consist of different independent parts, e.g. 2 main components can be:

1) "generator", "component for binary patch creation" - the program which creates binary patch as a whole or just adds additional service information to it; In some aspects, the "generator" also analyzes if the program can be patched. For example it checks that patch has a reference to a variable of the same size as it is in the binary. If patch references an object, then "generator" checks that it has the same structure. It can also check, that if patch references a constant variable, then it's defined as constant in the patch too. It can also check, that patch calls for a binary function with the right parameters. There can be much more sanity checks for variables and functions patch tries to reuse. In other aspects, binary patch can be created by other program, while generator adds additional service information to it (required by "patcher" to apply the patch, e.g. resolves external symbols) based on instrumentation made by human (in such case, generator analyzes whether the instrumentation was performed correctly, e.g. types and sizes of symbols, etc.). In some aspects, generator can also perform compilation of the patch (for example, by calling a compiler like gcc).

2) "patcher", "applier", "symbols resolver" - the program (or a set of programs) that applies binary patches.

Patcher should identify "when" and "where" to apply patch:

1) Patcher understands “When” to apply the patch using libunwind library (on the process stopped by ptrace/libcompel). This library provides the ability to identify whether the code we need to patch is executed now.

2) Patcher understands “Where” to apply the patch using process mappings (the patcher finds the target mapping (VMA)). After the target mapping was found, patcher maps the patch, resolves links to functions/variables and adds jump instructions into old (being patched) functions to new functions (in the patch)(or alike assembler instructions to transfer control). On the other hand, an exemplary aspect can be an "automatization" - set of solutions that allows us to automate creation and applying patches. So, in different exemplary aspects, there can be one automatic system or a set of tools for doing this.

How we create patch:

We do not compare binary code of already compiled versions. We rely on source code patches. (E.g. we rely on patch instrumentation-Another important difference with other solutions is our ability to patch the code for different processor architectures. Not only for x86-x64 as usual, but for lots of others. We can easily create patches for any architecture, because we do not compare binary code. To create a patch we use not binary code, but code written on high-level programming language (e.g. C, C++, any compiled language, etc. In case of Java, the binary on the disk should be patched previously.)

In Linux, there are patch series for source code (high level, not binary). Such patches should be applied before compilation of the program. This is our case, as we create a binary patch based on these source code patches.

1. We take patch written in a programming language, i.e. source code patch.
2. Applying it and copy modified/affected functions to another source file.
3. Modify it. Making used symbols global and external, this will help for the instrumentation of binary patch. Adding instrumentation if necessary (adding links on static variables and functions in original binary)
4. Compile. Create object file of shared library.
5. Convert instrumentation (e.g. by generator) into binary representation.

6. Apply the created patch (e.g. by patcher).

Initially, we have a source patch (a set of files with source code, which contain the difference between the source codes of 2 versions of the program (e.g. it even can be created by git "diff" command)). This code is usually created by human.

Then we use this patch to get new (updated with the patch) source files and copy changed functions to another source file, which will be used as patch source.

Then this code has to be compiled. But, for the purposes of binary patch creation, this source patch should be modified before compilation. If we try to compile the patch as is, we will very likely obtain compilation errors, because of symbols, which were not defined in the patch.

The modification includes at least the following: any symbols that are used in the source patch but are not defined there, will now be explicitly defined as external global symbols in modified patch. This concerns both the symbols that were defined in the initial program and the symbols that should be imported from libraries. This will help not only to compile the patch, but also to bind contexts of the program being patched and the applied patch (during patching). In some aspects, this modification is called source patch instrumentation and is performed using macroses.

Then we build the code so that the modified patch is compiled as a shared object (i.e. create object file of shared library). So, the binary patch appears.

Instrumentation information is then converted into binary representation by "generator".

Binary representation of instrumentation will help (during patching) for binding (e.g. creating links) the symbols (e.g. functions and variables) in the patch to the corresponding symbols in the program, which is patched (binding of the contexts).

So, we perform instrumentation of the source code, convert it into binary representation and add it to the executable file (e.g. .elf file). We add service information to the build packet (shared object). Symbols that were added while source code modification, i.e. symbols from original program also used in the patch, we will call "binding" symbols. For

“binding” symbols, their corresponding offsets in the original program (or library) are added to the binary patch (written to .elf).

So, the binary patch will contain additional metadata.

In some aspects, generator does the instrumentation (though it can be done as a step of binary patch creation program). In this case, after compilation, the patch and some information about it (about what exactly we patch: a library or not) are given to generator, which creates the binary patch.

In some aspects, Generator does not create patch: it “links” patch with original binary (adds additional information, used by patcher upon appliance to link the patch with these mentioned static functions and variables; i.e. generator generates service information, required to apply patch, but not the patch itself). Also generator checks types of symbols in patch and in binary, and also check whether the source code instrumentation was done correctly.

At this moment, the binary patch is ready to be applied.

In conventional art, instrumentation is not used. They just use the difference between 2 compiled versions. On contrary, we use the source code patch, compile sources with it (and instrumentation) applied. Then, the symbols in the patch are resolved using metadata, compiled out of instrumentation, added on source patch creation. Then binary patch is applied. And during this operation, the patcher rewrites offsets by real addresses of the symbols where needed.

Most of the operations performed for patch creation and appliance are architecture independent, so we can easily create patches for different architectures (e.g. x86, x64, arm, MIPS, S390, PPC64, aarch64, etc.) using the same method and even the same programs (e.g. only some compiler options should be different). Conventional art does not provide such opportunity.

One part of the program module, which is in charge of patch creation and applying, should be architecture dependent - it is a part, which performs old code modification to allow control transfer from the old code to the code in patch -. I.e. logic of the patcher depends on the architecture, because it modifies the binary code of the program being patched (e.g. put

“jump” instructions to offsets (or “call” or alike) to the beginning of the "old" functions in program code that should be replaced by the "new" functions from the patch).

Then the patching (applying of the patch) starts.

First, let's look at one of the important steps of patch applying, which relies on the fact that the binary patch was instrumented and that's why contains metadata,- resolving of the symbols used in the patch.(Symbols - a set of functions and values (static or dynamic) used in the binary, including those exported to and imported from other modules.) The program, which applies the patch, in some aspects, is called "patcher" (further, we will use this designation for the program module (e.g. a set of tools) which applies the patch). In one aspect, the patcher (or analogous module) sees the "binding" symbols in the patch as global and external; it also sees the metadata information (that was added by generator based on source code instrumentation) for these symbols. E.g., the corresponding offsets in original binary code (the code of the process we are patching) and the reference that these symbols belong to initial program and that there they were not global but static.

So, resolving the symbols is done as following: the patcher writes to the specified place (i.e. Global Offsets Table, GOT)in the patch the real address (in the process address space) of the symbol in the code of the program being patched. For “binding” symbols, the address is calculated using the address of the beginning of the program code and the offset to the symbol from the beginning of the code (which is known from the metadata added on during instrumentation). So, the patcher writes the addresses of the symbols to the specified place in the binary patch. (The generator writes - offsets from the beginning of the library, the patcher - real addresses in process address space). If there is no metadata information for some symbols, it means that these symbols are really external (they are not defined in the original program) and they should be imported from libraries. In such cases, the patcher behaves like the dynamic linker of the OS, i.e. finds the needed library (loads them to the process's address space if needed, i.e. if they were not loaded to it before) and finds the addresses of these symbols in the library export section.

Usually programs in user space use functions from dynamic libraries. To allow them to do this, dynamic linker resolves dependencies, it is done like following. We have a binary file,

which contains the list of needed functions from different libraries. Dynamic linker finds them and loads them to the process address space when needed. While loading the binary, dynamic linker wrote its own address instead of the addresses of functions from libraries that should be used by the binary later – dependencies. For example, when the first attempt to call "printf" occurs in binary, dynamic linker is called, and dynamic linker finds a library containing "printf" (according to some priority algorithm)- and writes the address of this function in the library to the special place in a binary.

We do not use the conventional dynamic linker (which is usually a part of the OS) to resolve symbols during patch applying, because "binding" symbols are internal for the program we patch, and the dynamic linker will either do not find anything with such names and fail, or will find something wrong and make the behavior of the patched program unpredictable. We can say that code in our binary is "position independent code (PIC)". In such code, in call command not the real address of a function is used, but the address of some place in the code where the real address of the function will be written. While loading dynamically linked binary, dynamic linker needs to find the address of the function and write it in some sell in memory of this binary. Then, all the references to this function will work automatically. But, there will be an additional jump on each calling of the function. All this provides an opportunity for using the method and system disclosed herein.

Now, let us look at the patching/applying operation as a whole.

The solution is to do as much as we can outside of the process, not inside of its context: to create a library containing a binary patch inside of it and to link this library to the process. The "patcher" works in user space, and patches user space processes. Patching kernel processes is easier, because everything is done in the same context, and the addresses of all functions and variables are known. In case of user space, from inside one process it is complicated to "see" addresses of functions in another process. Another problem is to map some binary into alien process address space. Patcher does it by stopping a process via ptrace/lbcompel and by injecting a binary blob, used later to mmap patch into process address space.

-

The “patcher” performs the following main steps (as shown on Fig. 2):

- 1) it redirects execution of the program (transfers control) from the old code to the new one,
- 2) links the new code and the data existing in the program (i.e. resolving "binding" symbols).

Redirecting.

If we want to change binary code of the program, we can put the new code fragment to the patch and map the patch, which looks like a library, to process's address space. Then “patcher” changes the old code to allow redirecting execution (transfer control) to the new code fragment in the patch. Here the new fragment should be written so that it returns execution to the proper instruction, after it ends execution. E.g., while applying the patch containing a new version of a function (or any code fragment), in the code of the program being patched we write “jump” instruction (or “call”, or alike) instead of the beginning of the corresponding old function (or into the place from which we should transfer execution to the code fragment in the patch).

Resolving symbols. One of the main problems is to link the new code and the data existing in the program. "Binding" symbols and binary code instrumentation help to solve this problem.

If there was a static symbol (e.g. function) to which we want to refer in the patch, we mark it in the patch as external and add additional information (a bit more code, wrapped as a macro) (at source code modification step, i.e. now it will be “binding” symbol) and then “generator” will convert it into binary representation, used later by patcher to apply the patch. E.g., compiler marks the symbol as global while generator adds additional info for “patcher” to execution file (e.g. .elf, .exe, etc.). The additional info is some metadata about the symbol, for example, the offset of this function in binary code of the program we patch. The “patcher” will calculate the real address of the corresponding static symbol in the program address space and write it to the patch. So all attempts to access this “binding” symbol from the patch will be redirected to the corresponding symbol, which exists in the program.

Generator links the symbols to their places in the binary file of the initial version of the program being patched and add this info to a binary patch. Then patcher computes real address

of such symbols using info put to the binary by generator. So, the part of the job for resolve symbols is made in generator, it is done to avoid repetitive computations and to reduce the time of applying the patch. E.g. If there are several processes then in each process the binary patch could be placed in different places. Patcher should resolve symbols in each of them. While process is stopped, the “patcher” (or in other aspects, it can be a module for resolving symbols):

- 1) resolves links to external symbols (i.e. global external symbols);

In some aspects, patcher does it during loading of the patch, though dynamic linker (the existing OS component for resolving dependencies) usually does it "on demand", i.e. when such symbol is accessed from the context of the process. All steps of patch applying: loading, initializing and resolving are done atomically from the slipping process's point of view.

- 2) resolves “binding” symbols (non-global (e.g. static, protected, hidden, internal) functions or variables that are used in patch and were defined in the program being patched). They were made external, using macroses during creation of the binary patch and then during instrumentation marked that they were static in the initial program.

Patcher works not inside of the process context, but outside (which is another difference from existing solutions). Patcher is started with special permissions/privileges (in some aspects, "root", because it will need to read proc, for example).

The steps of binary patch applying:

Let us assume that binary patch (a shared library) was already created. Binary patch creation was already described before. So to patch the process we need to:

- 1) link to the process via debugging interface;
- 2) stop the process and get inside of it (e.g. using ptrace-based interface; libCompel is one of the most comfortable, it even can perform the step 1));

At this step, we need to identify which part of the code is being executed now. If we stop the process while it is executing the code (or have this code in stack, i.e. it can return execution to it at some moment in time), which we want to change, we will not be able to change it, or otherwise the fault can occur. So, we should stop the process while it executes something we

do not want to change. Or we can stop the process, understand where we are, patch if we can, or release the process, if we cannot apply the patch, and then stop again. (See fig. 3).

3) map parasite code to the process;

4) transfer execution to parasite code, which loads the patch as a shared library (maps a patch to the process address space);

We load shared object file (also known as “*.so” file) from within the process by executing parasite code. It is mapped via “mmap” system call and then all its references are resolved (from outside by our “patcher”) without using dynamic linker (this is important).

In some aspects, parasite code is needed only for patch loading. But in some aspects, it is also used further in applying operation.

5) resolve external symbols used in the binary patch;

On this step "patcher" links libraries used in the patch (if they were not linked to the program previously), and resolves “binding” symbols in the patch.

6) the "patcher" amends the code of the running program to allow control transfer from the old code of the program to the new code in the patch;

At least, amend the process's binary code so that it started to use functions (or just some lines of the code, or symbols) from the patch instead of old ones.

At this step we need to switch process's context. This can be done by some code from inside the process, or from outside (e.g. by patcher). In some aspects, we do it from outside using ptrace, which allows for writing to any place of memory of the stopped process.

7) release (resume) the process; then the patched process runs.

As shown on the Fig.1, "patcher" performs the following operations:

- 1) Stop process; (step1 и 2)
- 2) Collect process VMA; (step2)
- 3) Check patch is applied; (step2)
- 4) Search VMA to patch;(step2)
- 5) Build process “needed SO” list;(step2)
- 6) Resolve patch relocations;(step2)

7) Apply patch; (step3-6)

8) Resume process. (step 7)

In some aspects, "Patcher" can have 2 modes:

1) via dbg inter face - just map everything to process memory (see above);

2) usingplag-in - libcdll open - creates socket, and sends everything via it. Plag-in is able to map and unmap a patch, and can send (upon request) the list of build process "needed SO" list (see step 5.)On Fig. 3 the steps of working of the exemplary "patcher" are shown.

The method and system improves computer functionality: the main is - reducing services downtime. I.e. some services can be on-the-fly patched instead of being restarted (e.g. data bases, restart of which is along and heavy operation).

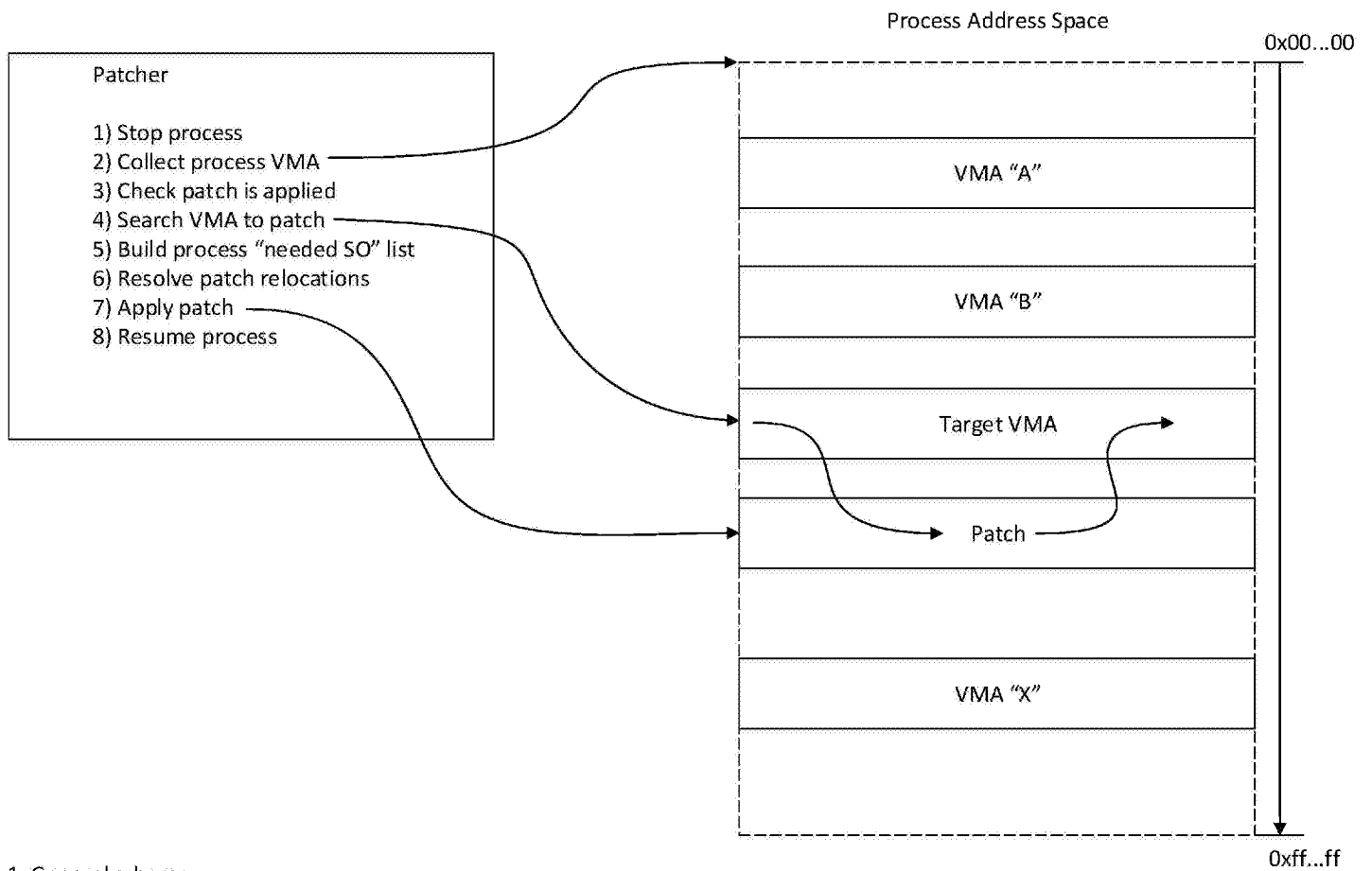


Fig. 1. General scheme

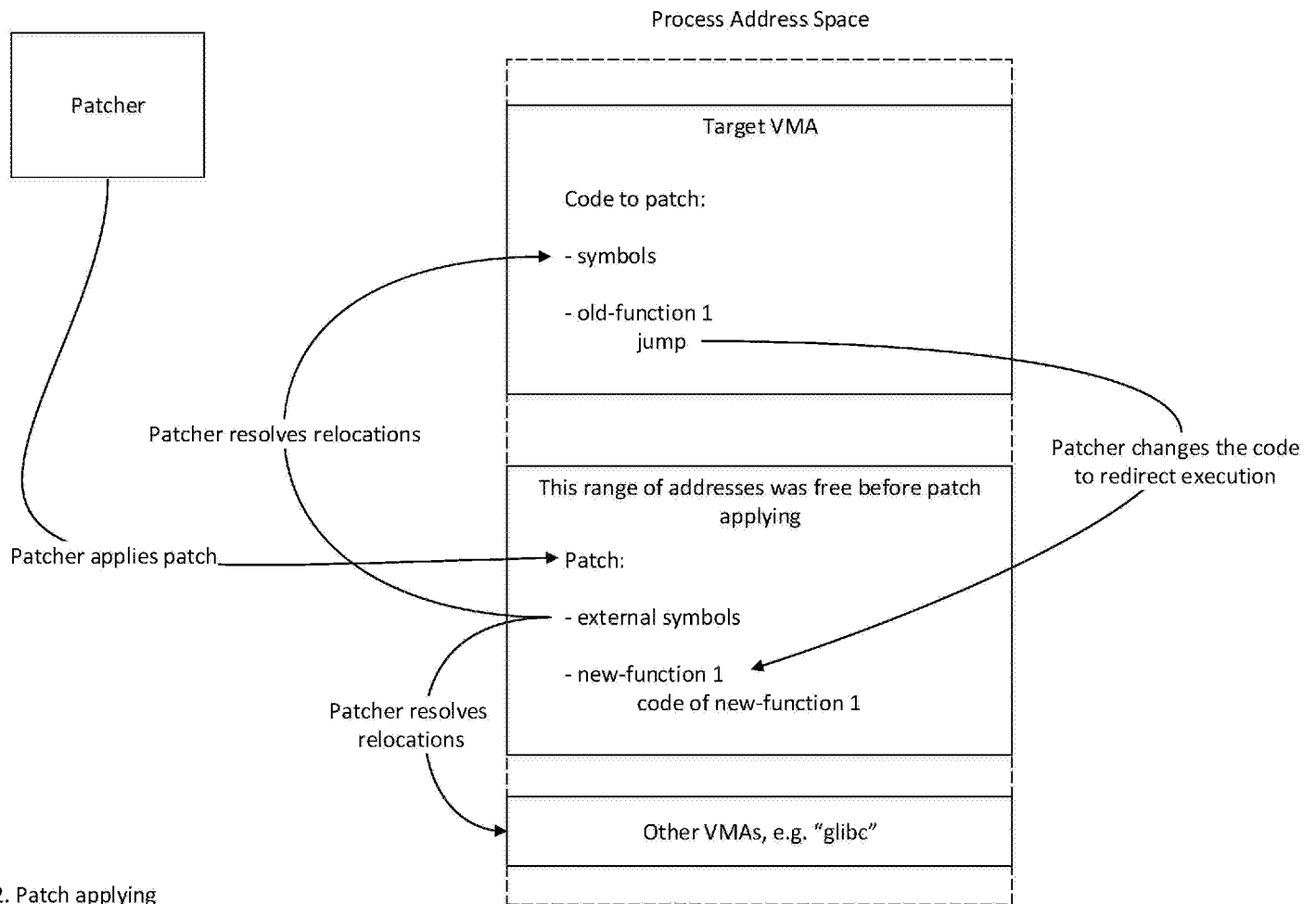


Fig. 2. Patch applying

Fig. 3. Steps of patcher operation ("patch" command)

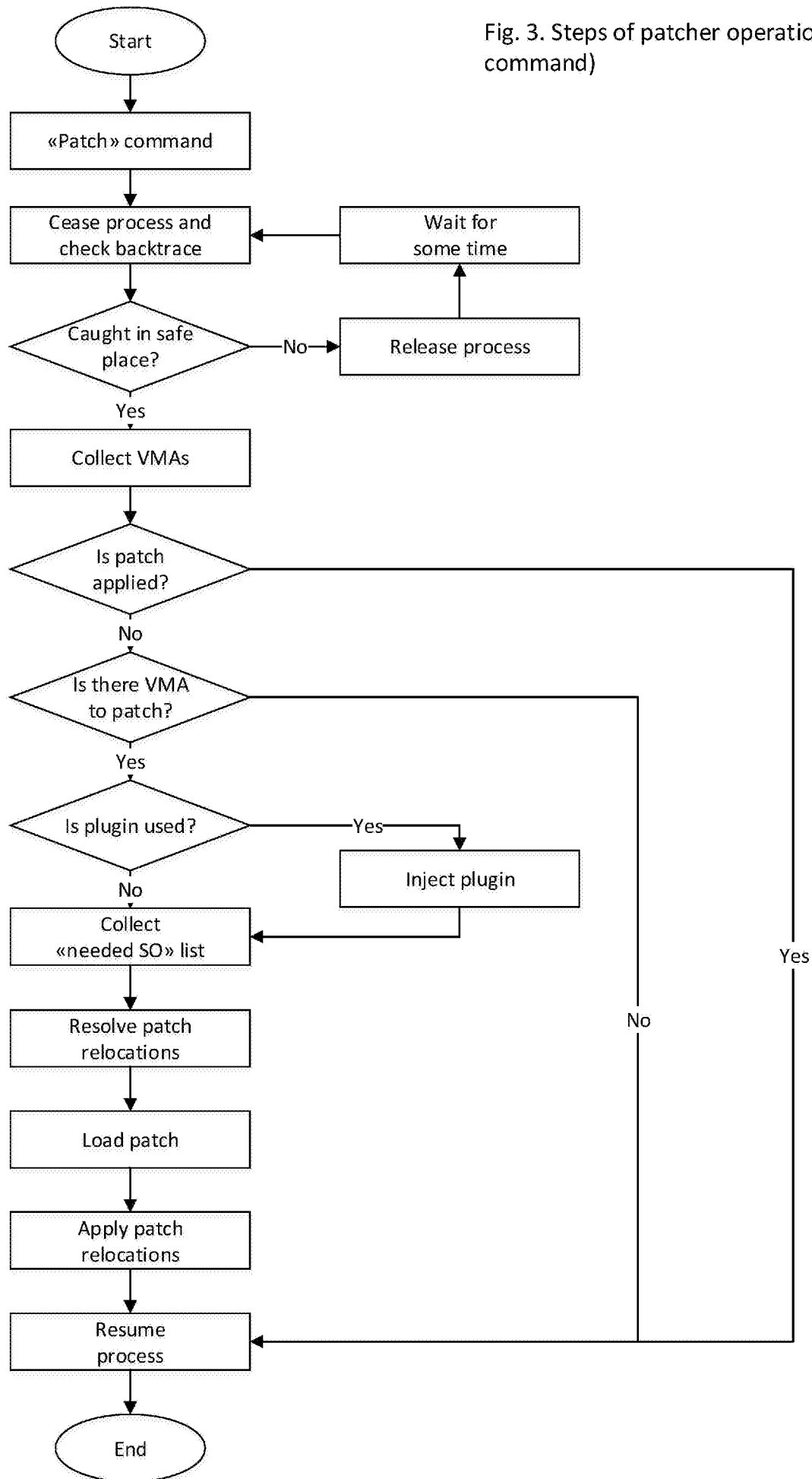
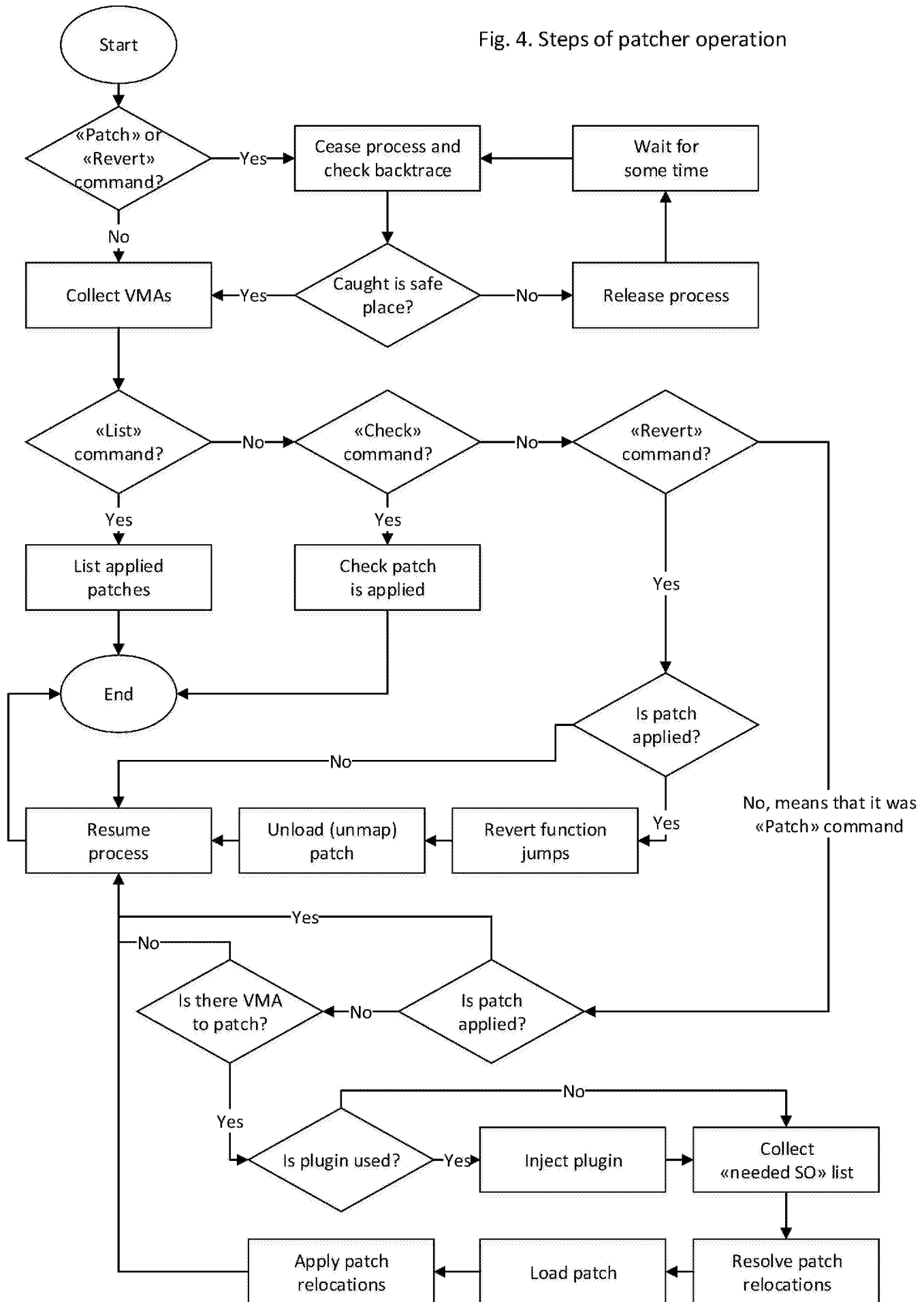


Fig. 4. Steps of patcher operation



Binary patch compiled as shared library

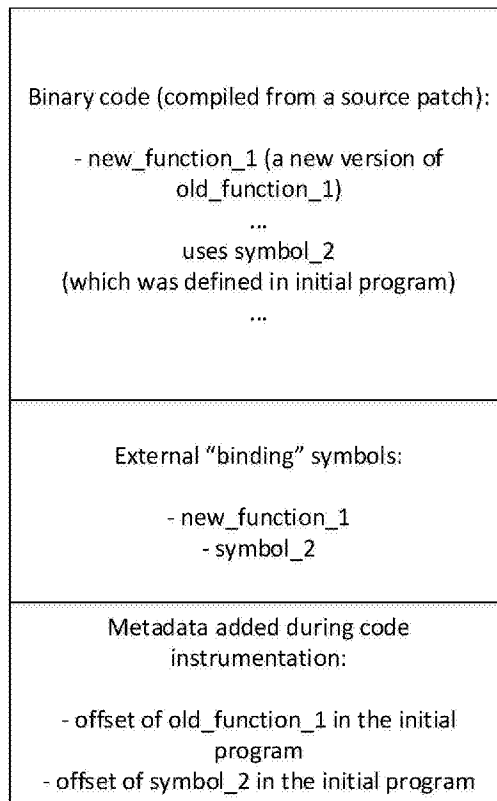


Fig. 5. Structure of the patch

Electronic Patent Application Fee Transmittal

Application Number:				
Filing Date:				
Title of Invention:	SYSTEM AND METHOD FOR USERSPACE LIVE PATCHING			
First Named Inventor/Applicant Name:	Stanislav Kinsburskiy			
Filer:	Michael Fainberg/erica jacobson			
Attorney Docket Number:	038179-00027			
Filed as Large Entity				
Filing Fees for Provisional				
Description	Fee Code	Quantity	Amount	Sub-Total in USD(\$)
Basic Filing:				
PROVISIONAL APPLICATION FILING	1005	1	260	260
Pages:				
Claims:				
Miscellaneous-Filing:				
Petition:				
Patent-Appeals-and-Interference:				
Post-Allowance-and-Post-Issuance:				

Description	Fee Code	Quantity	Amount	Sub-Total in USD(\$)
Extension-of-Time:				
Miscellaneous:				
Total in USD (\$)				260

Electronic Acknowledgement Receipt

EFS ID:	30837561
Application Number:	62580611
International Application Number:	
Confirmation Number:	1016
Title of Invention:	SYSTEM AND METHOD FOR USERSPACE LIVE PATCHING
First Named Inventor/Applicant Name:	Stanislav Kinsburskiy
Customer Number:	38485
Filer:	Michael Fainberg/erica jacobson
Filer Authorized By:	Michael Fainberg
Attorney Docket Number:	038179-00027
Receipt Date:	02-NOV-2017
Filing Date:	
Time Stamp:	14:09:41
Application Type:	Provisional

Payment information:

Submitted with Payment	yes
Payment Type	CARD
Payment was successfully received in RAM	\$260
RAM confirmation Number	110317INTEFSW14102700
Deposit Account	012300
Authorized User	Erica Jacobson

The Director of the USPTO is hereby authorized to charge indicated fees and credit any overpayment as follows:

37 CFR 1.21 (Miscellaneous fees and charges)

File Listing:

Document Number	Document Description	File Name	File Size(Bytes)/ Message Digest	Multi Part /.zip	Pages (if appl.)
1	Application Data Sheet	038179-00027ADS.pdf	7111763	no	7
			ac302c197fb1880a9bdc17f01fb83053c3de151		
Warnings:					
Information:					
This is not an USPTO supplied ADS fillable form					
2	Specification	038179-00027application.pdf	63659	no	12
			98f6ca99360dcc01e2c0523a78011f6c114cf02		
Warnings:					
Information:					
3	Drawings-only black and white line drawings	038179-00027drawings.pdf	1991275	no	5
			81dff9a2c0d9348dd0c0ba296a75881b3121d511		
Warnings:					
Information:					
4	Fee Worksheet (SB06)	fee-info.pdf	30214	no	2
			cf28b5068e66cea4d9bd10915f75201ea7ce0ba2		
Warnings:					
Information:					
Total Files Size (in bytes):			9196911		

This Acknowledgement Receipt evidences receipt on the noted date by the USPTO of the indicated documents, characterized by the applicant, and including page counts, where applicable. It serves as evidence of receipt similar to a Post Card, as described in MPEP 503.

New Applications Under 35 U.S.C. 111

If a new application is being filed and the application includes the necessary components for a filing date (see 37 CFR 1.53(b)-(d) and MPEP 506), a Filing Receipt (37 CFR 1.54) will be issued in due course and the date shown on this Acknowledgement Receipt will establish the filing date of the application.

National Stage of an International Application under 35 U.S.C. 371

If a timely submission to enter the national stage of an international application is compliant with the conditions of 35 U.S.C. 371 and other applicable requirements a Form PCT/DO/EO/903 indicating acceptance of the application as a national stage submission under 35 U.S.C. 371 will be issued in addition to the Filing Receipt, in due course.

New International Application Filed with the USPTO as a Receiving Office

If a new international application is being filed and the international application includes the necessary components for an international filing date (see PCT Article 11 and MPEP 1810), a Notification of the International Application Number and of the International Filing Date (Form PCT/RO/105) will be issued in due course, subject to prescriptions concerning national security, and the date shown on this Acknowledgement Receipt will establish the international filing date of the application.

EXHIBIT J

02/16/01
1041 U.S. PTO

This is a request for filing a PROVISIONAL APPLICATION FOR PATENT under 37 CFR 1.53(c)

Docket Number	44151-00006	Type a plus sign (+) inside this box ->	+
---------------	-------------	---	---

INVENTOR(S)/APPLICANT(S)			
LAST NAME	FIRST NAME	MIDDLE INITIAL	RESIDENCE (CITY AND EITHER STATE OR FOREIGN COUNTRY)
Tormasov	Alexander		Moscow, Russia
Lunev	Dennis		Moscow, Russia
Beloussov	Serguei		Singapore, Singapore
Protassov	Stanislav		Moscow, Russia
Pudgorodsky	Yuri		Moscow, Russia

TITLE OF THE INVENTION (280 characters max)			
USE OF VIRTUAL COMPUTING ENVIRONMENTS TO PROVIDE FULL INDEPENDENT OPERATING SYSTEM SERVICES ON A SINGLE HARDWARE NODE			

CORRESPONDENCE ADDRESS			
Alan R. Thiele JENKENS & GILCHRIST 1445 Ross Avenue, Suite 3200 Dallas, Texas 75202-2799			
STATE	Texas	ZIP CODE	75202-2799
		COUNTRY	US

ENCLOSED APPLICATION PARTS (check all that apply)			
<input checked="" type="checkbox"/> Specification	Number of Pages	10	<input checked="" type="checkbox"/> Small Entity Statement
<input checked="" type="checkbox"/> Drawing(s)	Number of Sheets	3	<input checked="" type="checkbox"/> Other (Specify) Express Mail Certificate

METHOD OF PAYMENT OF FILING FEES FOR THIS PROVISIONAL APPLICATION FOR PATENT (check one)		
<input checked="" type="checkbox"/> A check or money order is enclosed to cover the provisional filing fees		PROVISIONAL FILING FEE AMOUNT (\$)
<input type="checkbox"/> The Commissioner is hereby authorized to charge filing fees and credit Deposit Account Number:	10-0447	\$75.00

This invention was made by an agency of the United States Government or under contract with an agency of the United States Government



No.



Yes, the name of the U.S. Government/agency and the Government contract number are:

Respectfully submitted,

SIGNATURE

Date

2/16/00

TYPED or PRINTED NAME Alan R. Thiele

REGISTRATION NO.
(if appropriate)

30,694



Additional inventors are being named on separately numbered sheets attached hereto.

USE ONLY FOR FILING A PROVISIONAL APPLICATION FOR PATENT

Provisional Patent Application
Applicant: Alexander Tormasov, et al.
Attorney Docket No.: 44151.00006

**PROVISIONAL PATENT APPLICATION
OF
ALEXANDER TORMASOV,
DENIS LUNEV,
YURI PUDGORODSKY,
SERGEI BELOUSSOV,
AND
STANISLAV PROTASSOV**

ENTITLED:

**USE OF VIRTUAL COMPUTING ENVIRONMENTS TO PROVIDE
FULL INDEPENDENT OPERATING SYSTEM SERVICES ON A SINGLE
HARDWARE NODE**

CERTIFICATE OF MAILING 37 C.F.R. § 1.10

I hereby certify that this correspondence is being deposited with the United States Postal Service with sufficient postage as Express Mail, Express Mail No. EL525151963US, addressed to: BOX PROVISIONAL PATENT APPLICATION, Commissioner of Patents, Washington, D.C. 20231; on February 16, 2001.

Melanie Reese Capps

Printed Name of Person

Melanie Reese Capps
Signature

BACKGROUND OF THE INVENTION

Technical Field

This invention relates to providing full independent computer system services
5 across a network of remote computer connections.

Description of the Prior Art

The problem of providing computer services across remote computer connections
has existed during the last 30-40 years beginning with the early stages of computer
technologies. In the very beginning, during the mainframe computer age, this problem
10 was solved by renting computer terminals which belong to a mainframe and connected to
it via modem or dedicated lines and provide some mainframe access services. Later, with
the beginning of the age of personal computers and with the widespread acceptance of the
client-server model -- the problem of access to large information sources, at first look,
seems to have been solved -- everyone could have his own computer and then rent an
15 Internet connection to obtain access to information sources on other computers.

Today, with wide growth of Internet access, another problem has arisen -- the
problem of creation of the information sources. Usually users want to put out their own
information sources in the form of websites somewhere and provide other computer users
with access to these websites. However, it is not possible to install a web server on most
20 home connections to a personal computer, simply because usually the connection to the
network from a home computer is simply too weak. There is a big industry called a
"hosting service" which provides computer users with an ability to utilize installed web
services.

Usually when one wants to provide Internet users with some information, which information could be of interest for a wide range of Internet users (usually in web server form), one must both put the information somewhere, and one must also provide a reliable network connection to the information.

5 The problem of providing access by ordinary users to large capacity computers appeared practically at the moment of the beginning of their industrial production. In the epoch of the mainframes, when the access by all users directly to computer equipment was difficult, this access problem was solved by providing users with remote terminals. These remote terminals were used to obtain certain services from mainframe computers.

10 The advantage of using remote terminals with a mainframe computer was that the user had little trouble accessing both the mainframe computer hardware and to some extent accessing the software resident on the mainframe computer. This is because the administration of the operation of a mainframe computer has always dealt with the installation and updating of software.

15 Further on, in the epoch of personal computers, each user could gain access to computing power directly from his workplace or home. With the advent of access to the Internet, this situation could probably cover the needs of most all users for both large amounts of information and robust operating systems.

20 The client-server model of networking computers provides an access system in which a personal computer is designated as the client computer and another computer or a set of computers is designated as the server computer -- access to which is carried out in a remote way covering the majority of needs of the common computer users.

But even the client-server model has some very fundamental drawbacks. Specifically, the high price of servicing of many client workplace computers, which includes creating a network infrastructure and the installation and the upgrading of software and hardware to obtain the bandwidth of network access for client computers, is a significant drawback. Additionally, the rapid growth of information available worldwide on the Internet has produced more users who in turn place even more information on the Internet. The required service to client computers should be provided by a server computer (most often it is web or www server) which, by itself, is powerful enough and is provided with an access channel to the Internet with corresponding power. Usually, personal computers of users have enough performance capability to interact with most of the web servers, but the typical channel of access to the network is most often one or two orders less productive than is needed. Besides, most often the personal computers in most homes cannot provide a sufficient level of service reliability and good level of security, etc. The same problems, besides the Internet services, appear for common users when very complex software packages are used. Users spend a lot of effort setting up and administering these complex software packages. To solve the given problems with web service, the technologies of using remote web hosting are usually applied wherein some company (usually ISP Internet Service Providers) provide the services of hosting of the web servers for the users. In that case the user is restricted to utilization of the standard preinstalled web server of the ISP. As a result the user is restricted in his possibilities.

Problems usually arise with the use of CGI scripts and more complex applications requiring, for example, a data base. Such computer tools can't be used for access to any

program of the user set on a remote server. The reason is that the user has become used to the absolute freedom in the adjustment of his local machine. But, the limitations that are imposed upon a user by the administration of the remote node are often unacceptable to a user.

5 The reasonable solution of such problem is by the use of emulators of computers. The operating system for IBM mainframe computers is named OS/390 and has been used for many years. Each user is given his own fully-functional virtual computer with emulated hardware. The implementation of the disclosed approach is connected with high costs, as every operating system installed in the corresponding virtual computer does
10 not know anything about the existence of the neighboring analogous computers and shares practically no resources with neighboring computers. Experience has shown that the price associated with virtual computers is very great.

Another analogous solution for non-mainframe computers utilizes software emulators of the VMware type. The software programs of the VMware type exist for
15 different types of operating systems and wholly emulate some typical computer inside one process of the main computer operating system.

The main problem is that not many of such computer emulators on a server having a typical configuration can be used. Usually the limitations on the use of computer emulators are connected with the fact that the size of the emulated memory is
20 close to the size of the memory used by the process or in which the computer emulator works. That is, the number of computer emulators that can be used on one server simultaneously is confined to a numeric range from 2-3 to 10-15. All the above mentioned solutions could be classified as multikernel implementations of virtual

computers when on one physical computer there are simultaneously several kernels of operating systems that are unaware of each other.

So, when it is necessary for many users to deal with a hosting computer, there is a necessity to provide each user with a complete set of services that the user can expect from the computer (the provision of virtual environments emulating with maximal completeness of the whole computer with the operating system installed in it). For the purpose of effectiveness of the use of equipment, it is desirable that the number of such computers in a virtual environment installed in one computer should be two or three orders more than in the above mentioned cases.

10 **BRIEF SUMMARY OF THE INVENTION**

The present invention describes a method of efficient utilization of a single hardware system with a single Operating System kernel. The end user is provided with a virtual computing environment that is functionally equivalent to a computer with a full-featured Operating System. There is no emulation of hardware or dedicated physical memory or any other hardware resources as in the case of solution of VMware kind.

The method of the present invention is implemented by the separation of user processes on the level of namespace and on the basis of restrictions inside the Operating System kernel. Virtual computing environment processes are never visible to other virtual computing environments running on the same computer. A virtual computing environment root file system is also never visible to other virtual computing environments running on the same computer. The root file system of a virtual computing environment is working in the read-write mode and allows the root user of every virtual computing environment to perform file modifications and Operating System configuration.

BRIEF DESCRIPTION OF THE SEVERAL VIEWS OF THE DRAWING

A better understanding of the present invention may be had by reference to the drawing figures, wherein:

5 FIG. 1 shows a network of end users with access to virtual computing environments encapsulated in a computer with a full feature operating system in accordance with the present invention;

 FIG. 2 shows a utilization of resources of hardware (memory and file system) by different virtual computing environments; and

10 FIG. 3 shows a utilization of resources of hardware (memory and file system) in another full hardware emulation solution.

DETAILED DESCRIPTION OF THE INVENTION

 The disclosed invention presents the method of efficient utilization of a single
15 hardware system with a single Operating System kernel.

 The goal of the present invention is to provide the possibility of work on a system that is the functional equivalent of a computer with a full-featured Operating system.

 As a result the possibility appears to the user as if he has obtained full network root access to a common computer with fully-featured Operating System installed on it.

20 Specifically, the end user is provided with a virtual computing environment that is functionally equivalent to a computer with full-featured Operating System.

 From the point of view of the end user, each virtual computing environment is the actual remote computer with the network address in which the end user can perform all actions allowed for the ordinary computer: the work in command shells, compilation and
25 installation of programs, configuration of network services, offices and other

applications. Several different users can work with the same hardware node without noticing each other in the same way as if they worked on different computers (Fig. 1).

Each virtual computing environment includes a complete set of processes and files of an operating system that can be modified by the end user.

5 In addition the end user may stop and start the virtual computing environment in the same way that is done with a common operating system. Nevertheless all virtual computing environments share one and the same kernel of the operating system. All the processes inside the virtual computing environment are the common processes of the operating system and all the resources inherent to each virtual computing environment
10 are shared in the same way as typically happens inside an ordinary single kernel operating system.

Fig. 2 shows the process of the coexistence of the two virtual computing environments on one hardware computer. Each virtual computing environment has its own unique file system and each virtual environment can also see the common file
15 system. All the processes of all virtual computing environments work from inside the same physical memory. If two processes in different virtual computing environments were started for execution from one file (for example from the shared file system) they would be completely isolated from each other.

In such a way the high effectiveness of implementation of multiple virtual
20 computing environments inside one operating system is reached.

There is no emulation of hardware or dedicated physical memory or another hardware resource.

The disclosed invention differs from the other solutions (Fig. 3) that provide a complete emulation of computer hardware and give the user the full scope virtual
25 computer with the doubled expenses. This happens because a minimum of 2 actual

kernels are performed in the computer, one inside the other - the kernel of the main operating system and inside the process, the kernel of the emulated operating system.

The implementation of the kernel of the Operating System with the properties necessary for this invention should carry out the separation of the users not on the level of
5 hardware but on the level of the namespace and on the basis of limitations implemented inside the kernel of the Operating System.

Virtual computing environment processes are never visible to other virtual computing environments running on the same computer. The virtual computing environment root file system is also never visible to other virtual computing
10 environments running on the same computer. The root file system of the virtual computing environment is working in the read-write mode and allows a root user of every virtual computing environment to make file modifications and configure the operating system.

The changes done in the file system in one virtual computing environment do not
15 influence the file systems in the other virtual computing environment.

CLAIMS

We claim:

1 1. A method for efficient utilization of a single hardware system with a
2 single operating system kernel including a virtual computing environment functionally
3 equivalent to computer having a full-featured operating system is provided to an end user
4 without emulation of hardware or dedicated physical memory or hardware resource, and
5 where such method is realized by separation of user processes on the level of namespace
6 and on the base of restrictions implemented inside said operating system kernel.

1 2. The method as defined in Claim 1 wherein each virtual computing
2 environment is not visible to other virtual computing environments on the system.

1 3. The method as defined in Claim 1 wherein each virtual computing
2 environment has a completely independent root file system.

ABSTRACT OF THE DISCLOSURE

Method of efficient utilization of a single hardware system with single Operating System kernel wherein a Virtual Environment, functionally equivalent to full-featured Operating System box, is provided to an end user without emulation of hardware or dedicated physical memory or another hardware resource. Such method is realized by separation of user processes on the level of namespace and on the basis of restrictions implemented inside the Operating system kernel. Each Virtual Environment is not visible to another Virtual Environment within the system and has a completely independent root file system.

The work of end users with virtual environments
encapsulated in the same hardware box

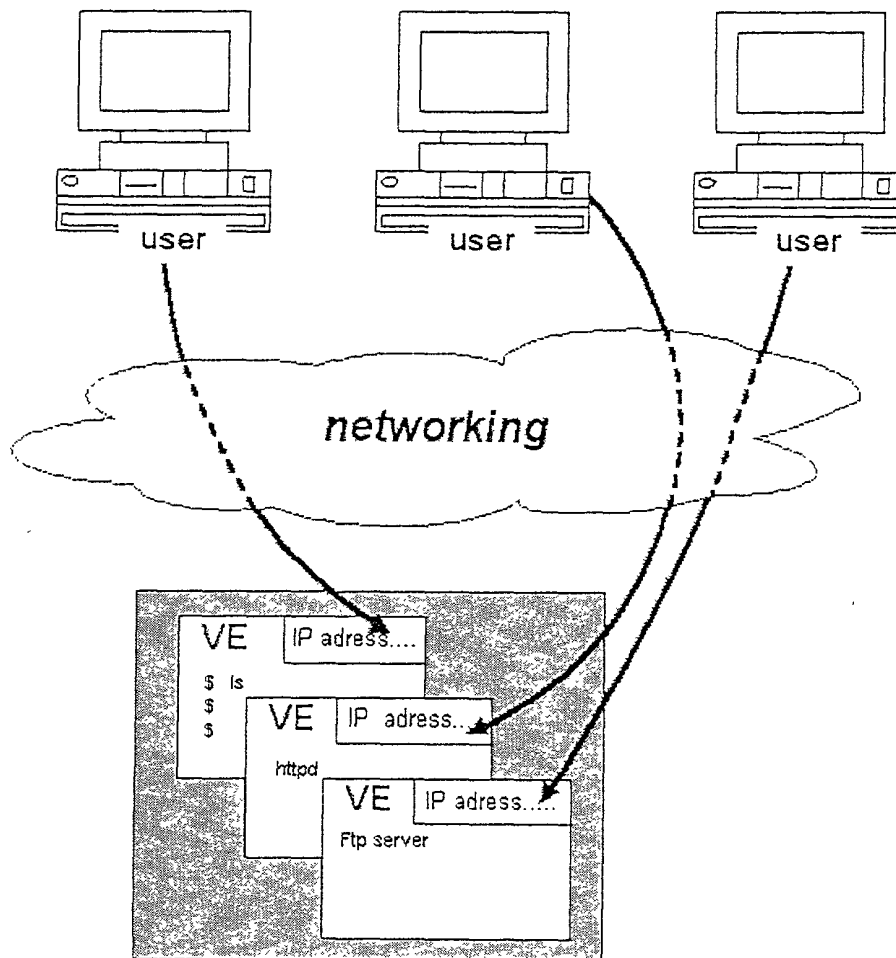


FIG 1

Utilization of resources of hardware (memory and file system) by different Virtual Environment's

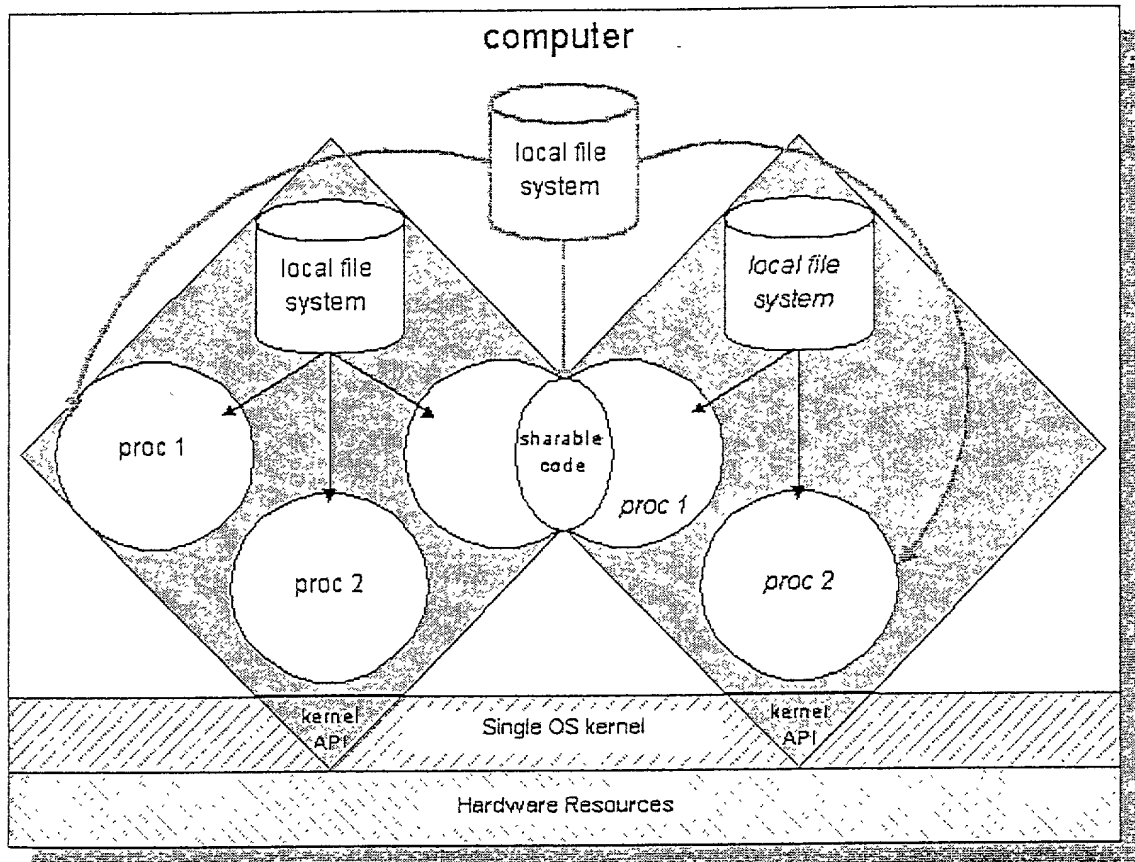


FIG 2

Utilization of resources of hardware (memory and file system) in another full hardware emulation solutions

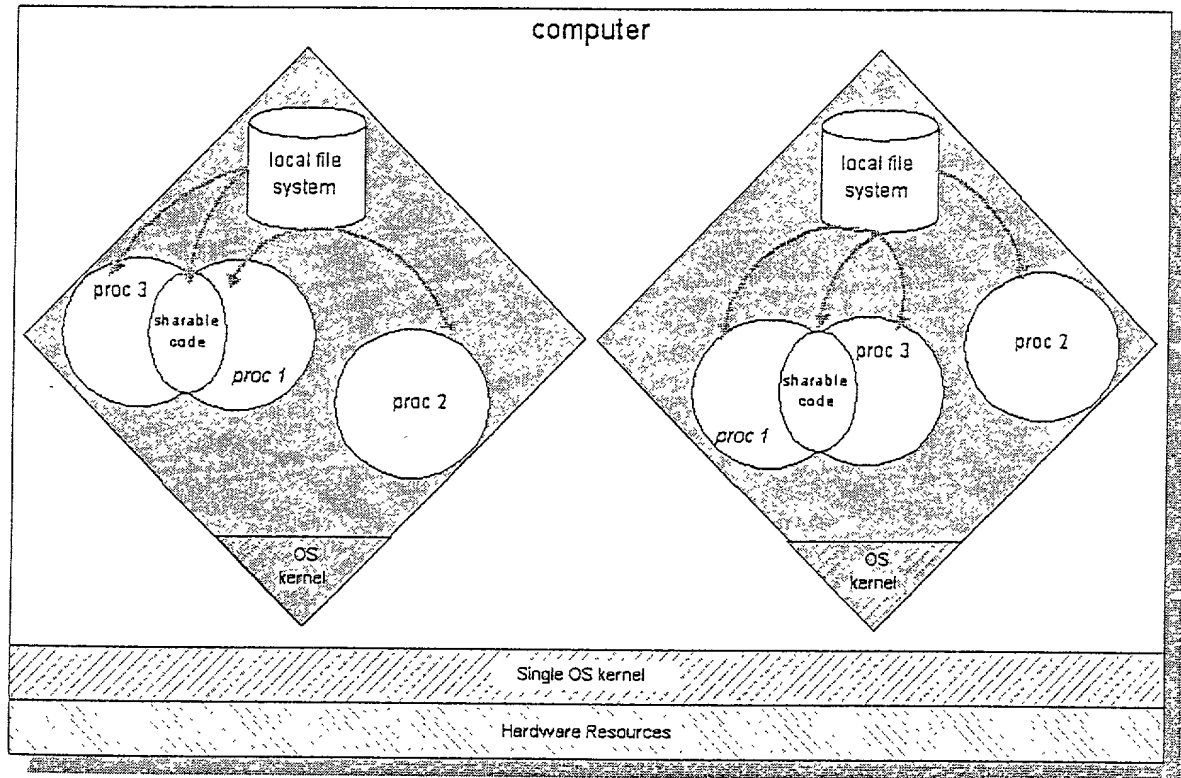
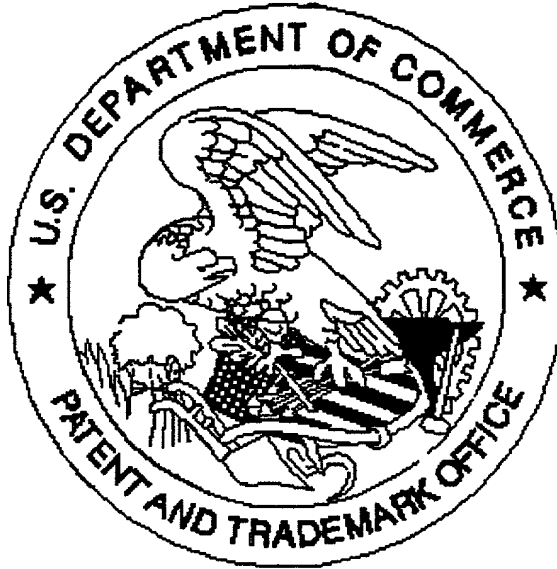


FIG 3

United States Patent & Trademark Office
Office of Initial Patent Examination -- Scanning Division



Application deficiencies found during scanning:

☐ Page(s) _____ of small entity statement were not present
for scanning. (Document title)

☐ Page(s) _____ of _____ were not present
for scanning. (Document title)

☐ *Scanned copy is best available.*

EXHIBIT K

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re application of:

PROTASSOV *et al*

Appl. No.: 11/757,598

Filed: June 4, 2007

For: **SYSTEM AND METHOD FOR
MANAGEMENT OF VIRTUAL
EXECUTION ENVIRONMENT DISK
STORAGE**

Confirmation No.: 2963

Art Unit: 2186

Examiner: MERCADO, RAMON A.

Atty. Docket: 2230.0850001

Amendment and Reply Under 37 C.F.R. § 1.111

Mail Stop Amendment

Commissioner for Patents
PO Box 1450
Alexandria, VA 22313-1450

Sir:

In reply to the Office Action dated **May 20, 2011**, Applicants submit the following
Amendment and Remarks. This Amendment is provided in the following format:

- (A) Each section begins on a separate sheet;
- (B) Starting on a separate sheet, a complete listing of all of the claims:
 - in ascending order;
 - with status identifiers; and
 - with markings in the currently amended claims;
- (C) Starting on a separate sheet, the Remarks.

It is not believed that extensions of time or fees for net addition of claims are required
beyond those that may otherwise be provided for in documents accompanying this paper.

However, if additional extensions of time are necessary to prevent abandonment of this

- 2 -

Protassov *et al*
Appl. No. 11/757,598

application, then such extensions of time are hereby petitioned under 37 C.F.R. § 1.136(a), and any fees required therefor (including fees for net addition of claims) are hereby authorized to be charged to our Deposit Account No. 50-3523.

Atty. Dkt. No. 2230.0850001

Amendments to the Claims

The listing of claims will replace all prior versions, and listings of claims in the application.

1. (currently amended) A method for storing data of a Virtual Execution Environment (VEE), comprising:

launching, on a computer system, isolated VEEs with a shared OS kernel;
starting a common storage device driver and a common file system driver;
mounting a virtual disk drive for each VEE, wherein each VEE has its own file system driver object for handling a disk image of its VEE;
wherein the virtual disk drive is represented on the storage device as a disk image, and
wherein the disk image is stored on a physical storage device or on a network storage device as at least one file; and

wherein the file of the virtual drive comprises an internal structure that includes user data storage blocks and redirection blocks that point to user data storage blocks and that are used by the VEE's file system driver.

2. (original) The method of claim 1, wherein the file is a generic file whose size is dynamically changed when content of the virtual disk drive is changed.

3. (canceled)

4. (currently amended) The method of claim [[3]] 1, wherein the at least one file has a B-tree or B+tree structure.

5. (currently amended) The method of claim [[3]] 1, wherein the redirection blocks have a ~~multilevel~~ multi-level hierarchy.

6. (currently amended) The method of claim 1, wherein the VEEs' file system drivers and the common driver supporting the virtual drive supporting means ~~is~~ are running in operating system space and ~~is~~ are used by different VEEs.

7. (original) The method of claim 1, wherein the disk image further comprises files common to different VEEs.

8. (original) The method of claim 1, wherein the disk image files are dynamically updated during writes to the virtual disk drive.

9. (original) The method of claim 1, wherein the disk image files are reduced in size to exclude data corresponding to unused blocks of the virtual disk drive.

10. (original) The method of claim 1, wherein the disk image files contain only used blocks data.

11. (original) The method of claim 1, wherein some disk image files can be shared between VEEs.

12. (original) The method of claim 1, wherein the drive image files support starting the VEE contents on another hardware system or other VEE.

13. (currently amended) The method of claim 1, wherein ~~the~~ an internal structure of the at least one file ~~provide possibility of~~ that allows for writes to the disk image in a fault tolerant manner; and

any virtual disk writes are added to the at least one file with preserving the internal structure integrity in any moment of time.

14. (original) The method of claim 1, further comprising online snapshotting of the virtual disk drive using an internal structure of the at least one file.

15. (original) The method of claim 1, further comprising offline snapshotting of the virtual disk drive using an internal structure of the at least one file.

16. (currently amended) A method for storing data of a Virtual Machine, comprising:
starting an operating system running a computing system;
starting a Virtual Machine Monitor (VMM) under control of the operating system, wherein the VMM virtualizes the computing system and has privileges as high as the operating system;

creating isolated Virtual Machines (VMs), running on the computing system simultaneously, wherein each VM executes its own OS kernel and each VM runs under the control of the VMM;

starting a storage device driver and a file system driver in the operating system;

mounting a virtual disk drive;

starting VM-specific file system drivers in the ~~VM~~ VMs,

wherein the VM specific file system driver together with ~~the~~ common storage device drivers support the virtual disk ~~drives~~ drive,

wherein the virtual disk drive is represented on the storage device as a disk image,

wherein the disk image data are stored on the storage device as at least one file that includes user data storage blocks and redirection blocks, the redirection blocks point to user data storage blocks,

wherein the redirection blocks have a multilevel hierarchy, and

~~the~~ an internal structure is used by the VM-specific file system driver.

17. (original) The method of claim 16, wherein each VM uses a common storage device driver and a common file system driver.

18. (currently amended) A method for storing data of a Virtual Machine, comprising:
starting a Virtual Machine Monitor (VMM) running with system level privileges;

creating a primary Virtual Machine (VM) without system level privileges and having a host operating system (HOS) within it, the HOS having direct access to at least some I/O devices;

creating a secondary Virtual Machine running without system level privileges;

starting a storage device driver and a file system driver in the HOS;

starting a VM-specific file system driver in the secondary VM;

starting a VM-specific virtual device driver in the secondary VM that provides access to the virtual storage drive,

wherein the VM specific virtual device driver has access to the disk image files from the a virtual device driver via the file system driver and the storage device driver; and

mounting a virtual disk drive,

wherein the virtual disk drive is represented on a storage device as a disk image, and

wherein the disk image data is stored on the storage device as at least one file; and

wherein the disk image comprises an internal structure that includes user data storage blocks and redirection blocks that point to user data storage blocks and that are used by the VM-specific file system driver.

19. (currently amended) A method for storing data of Virtual Execution Environments, comprising:

starting a Virtual Machine Monitor (VMM) with system level privileges;

creating a primary Virtual Machine (VM) without system level privileges and having a host operating system (HOS) running within it;

starting, a safe interface providing secure access from the VMs to computing system hardware;

creating a secondary Virtual Machine running without system level privilege, wherein the secondary VM uses the safe interface for accessing hardware;

starting a storage device driver and a file system driver in the HOS;

starting a VM-specific file system driver in the secondary VM;

starting a VM-specific virtual device driver in the secondary VM that provides access to ~~the~~ a virtual storage drive and has access to ~~the~~ disk image files from the VM-specific virtual device driver via the file system driver, the storage device driver and the safe interface; and

mounting a virtual disk drive for the secondary VM;

wherein the virtual disk drive is represented on a storage device as a disk image; and

the disk image data is stored on the storage device as at least one file; and

wherein the disk image comprises an internal structure that includes user data storage blocks and redirection blocks that point to user data storage blocks and that are used by the VM-specific file system driver.

20. (currently amended) A method for storing data of Virtual Execution Environments, comprising:

starting an operating system on a computing system that uses a system level file system cache for caching data being read or written to a storage device;

starting a Virtual Machine Monitor (VMM) under the control of the operating system;

starting isolated Virtual Machines (VMs) on the computing system, wherein each VM executes its own OS kernel and each VM runs under the control of the VMM;

starting a common storage device driver and a common file system driver in the operating system;

mounting a virtual disk drive, the virtual disk drive provides correct functioning of the ~~VM~~ VMs;

starting ~~[[a]]~~ VM-specific file system drivers in the ~~VM~~ VMs that, jointly with the common ~~drivers~~ driver, supports a virtual disk drive that is represented on the storage device as a disk image;

wherein the disk image data is stored on the storage device as at least one file;

wherein writes to the at least one file are executed with bypassing the file system cache;

wherein ~~the~~ an internal structure of the file enables writes to the disk image in a fault tolerant manner; and

wherein any virtual disk writes are added to the file while preserving the internal structure integrity of the file at any moment of time; and

wherein the internal structure includes user data storage blocks and redirection blocks that point to user data storage blocks and that are used by the VM-specific file system driver.

21. (currently amended) A method for storing data of Virtual Private Servers (VPSs), comprising:

starting an operating system on a computing system;

initiating a plurality of VPSs on the computing system, all the VPSs sharing a single instance of the operating system;

allocating, to each VPS, a dynamic virtual partition, wherein the size of the dynamic virtual partition changes in real time as VPS storage requirements change; and

wherein the dynamic virtual partition contains used and unused data blocks.

22. (currently amended) A computer program product for storing data of Virtual Machines (VMs), the computer program product comprising a non-transitory computer useable medium having computer program logic stored thereon for executing on at least one processor, the computer program logic comprising:

computer program code means for starting an operating system on a computing system;

computer program code means for initiating a Hypervisor having full system privileges;

computer program code means for initiating a plurality of VMs on a single OS, the VMs being managed by a Virtual Machine Monitor, wherein at least one of the VMs is running a Service OS; and

computer program code means for generating a dynamic virtual partition allocated to each VM, wherein the size of the dynamic virtual partition changes in real time as VM storage requirements change, and

wherein the dynamic virtual partition contains used and unused data blocks.

Remarks

Reconsideration of this Application is respectfully requested.

Upon entry of the foregoing amendment, claims 1, 2, 4 - 22 are pending in this application. Claims 1, 4 - 6, 13, 16, 18, 19, 20, 21 and 22 are amended. Claim 3 is canceled. These changes are believed to introduce no new matter, and their entry is respectfully requested.

Claim 22 is rejected under 35 U.S.C. § 101 as being allegedly directed to non-statutory subject matter. Claims 6, 13, 16 - 20 are rejected under 35 U.S.C. § 112 (second paragraph) as being allegedly indefinite. Claims 1-2, 6-8, 11-15, 21 and 22 stand rejected under 35 U.S.C. § 102(b) as being allegedly anticipated by Warren (The VMware Workstation 5 Handbook, 2005). Claims 3, 5, 16 - 18 stand rejected under 35 U.S.C. § 103(a) as being allegedly unpatentable over Warren (The VMware Workstation 5 Handbook, 2005). Claims 4, 9 and 10 stand rejected under 35 U.S.C. § 103(a) as being allegedly unpatentable over Warren (The VMware Workstation 5 Handbook, 2005) in view of Karpoff et al. (US Patent No. 6,875,059 B2). Claim 19 stands rejected under 35 U.S.C. § 103(a) as being allegedly unpatentable over Warren (The VMware Workstation 5 Handbook, 2005) in view of Goldsmith (US 2006/0069828A1). Claim 20 stands rejected under 35 U.S.C. § 103(a) as being allegedly unpatentable over Warren (The VMware Workstation 5 Handbook, 2005) in view of Kim et al. (IEEE ICME 2001).

Based on the above amendment and the following remarks, Applicants respectfully requests that the Examiner reconsider all outstanding rejections and that they be withdrawn.

Rejections under 35 U.S.C. § 101

Claim 22 has been amended to recite “a non-transitory computer useable medium.”

Thus, the claim rejection has been overcome and withdrawal of the rejection is requested.

Rejections under 35 U.S.C. § 112

Claim 6 has been amended to recite “VEEs’ file system drivers and the common driver supporting the virtual drive are running in operating system space and are used by different VEEs” to correct insufficient antecedent basis. Claim 13 has been amended to recite “an internal structure” to correct insufficient antecedent basis. Claim 13 has been further amended to recite “at least one file that allows for writes to the disk image” to clearly point out what is included or excluded by the claim language. Claim 16 has been amended to recite “starting VM-specific file system drivers in the VMs” to clarify that multiple Virtual Machines are used. Claim 16 has been further amended to recite “file system driver together with common storage device drivers” to correct insufficient antecedent basis. Claim 16 has been yet further amended to recite “an internal structure” to correct insufficient antecedent basis.

Claim 18 has been amended to recite “the VM specific virtual device driver has access to disk image files from the VM specific virtual device driver” to correct insufficient antecedent basis. Claim 19 has been amended to recite “a virtual storage drive and has access to disk image files from the VM-specific virtual device driver via the file system driver, the storage device driver and the safe interface” to correct insufficient antecedent basis and to eliminate multiple possible interpretations of the claim language. Claim 20 has been amended to recite “starting VM-specific file system drivers in the VMs” to clarify that multiple Virtual Machines are used.

Claim 20 has been further amended to recite “jointly with the common driver” to correct insufficient antecedent basis. Claim 20 has been yet further amended to recite “an internal structure of the file” to correct insufficient antecedent basis.

Thus, the claim rejections have been addressed and withdrawal of the § 112 rejections is requested.

Rejections under 35 U.S.C. § 102(b)

Claims 1-2, 6-8, 11-15, 21 and 22 stand rejected based on Warren. Although Applicants do not necessarily agree with the reasoning expressed in the Office Action, Applicants have amended the claims to recite additional aspects. Independent claims 1, 21 and 22 have been amended to recite “the file of the virtual drive comprises an internal structure that includes **user data storage blocks** and **redirection blocks** that point to user data storage blocks that are used by the VEE’s file system driver” and to recite that “the dynamic virtual partition contains **used** and **unused data blocks**”, respectively. These amendments emphasize that the claimed combination uses a two-level structure of a VM (VEE). This structure is implemented as a virtual partition used by the VM (VEE), which stores only user blocks (i.e., used data blocks). It also has a structure that describes (references) the used/unused data blocks.

Warren does not teach an internal structure that includes user data storage blocks and redirection blocks

Warren teaches a system where multiple operating systems run on a single PC (see page 2). Warren also teaches creating a Virtual Machine (VM) with a Host Computer (see page 76). Furthermore, Warren teaches configuration files for VM images (see table 5.1, pages 74-76). However, Warren does not teach or suggest that the files include an internal structure that “that

includes **user data storage blocks** and **redirection blocks** that point to user data storage blocks that are used by the VEE-specific file system driver”, unlike what is claimed. Instead, Warren merely uses lock files and log files for preventing and tracking changes to the VM. This is a process that the claimed combination is intended to improve upon by separating the user data blocks and VM (VEE) data blocks.

Warren does not teach a shared OS kernel

Independent claims 1, 21 and 22 (as amended) recite “isolated VEEs with a shared OS kernel” and “VPSs sharing a single instance of the operating system”, respectively. Warren specifically teaches creating a VM on a host computer (see page 76, and Figs. 5.2 – 5.4). Warren does not create multiple VMs that share the same OS. A canonical VM (as described in Warren) is a fully isolated entity that is unaware of anything other than itself. It is unaware of the existence of the host OS, and cannot share the host OS – there are no APIs that permit a canonical VM to request anything from the host. What is claimed – VEEs that share the same kernel – is a different virtualization scheme than what Warren teaches.

When multiple VM/VEEs share the same OS, the data management becomes a critical issue. Redundant data can be stored or some critical data can be lost. These issues are addressed by the claimed combination where a two-level structure of a VM (VEE) is used. This structure is implemented as a virtual partition used by the VM (VEE), which stores only user blocks (i.e., used data blocks). It also has a structure that describes (references) the used/unused data blocks. None of these features are needed for a single VM of Warren. Therefore, claims 1, 21 and 22 are not anticipated by Warren for these additional reasons.

Thus, Warren does not teach the above mentioned claimed features inherently or explicitly. Therefore, claims 1, 21 and 22 are not anticipated by Warren.

Rejections under 35 U.S.C. § 103(a)

Claims 3, 5, 16-18 stand rejected as being allegedly obvious over Warren. Claim 3 has been canceled and the language of claim 3 has been incorporated into the independent claim 18, which now recites that “the disk image comprises an internal structure that includes user data storage blocks and redirection blocks that point to user data storage blocks and that are used by the VM-specific file system driver.” Claim 16 recites “the disk image data are stored on the storage device as at least one file that includes user data storage blocks and redirection blocks, the redirection blocks point to user data storage blocks.” With regard to these features, the Office Action relies on page 12 of Warren where Linux OS is mentioned. The Office Action states that Linux has a specific directory structure to support multiple users, where each user has their own allocated directory under the home system directory. Applicants respectfully disagree and point out that Linux is completely different from what is claimed.

The claimed combination uses a two-level structure of a VM (VEE). This structure is implemented as a virtual partition used by the VM (VEE), which stores only user blocks (i.e., used data blocks). It also has a structure that describes (references) the used/un-used data blocks. Instead, Linux merely uses home system directory and user configuration files. The claimed combination adds an additional structure - a partition which only stores useful information and redirects. The system does not store un-used blocks. VM (VEE) sees the virtual partition with its local guest file system mounted on it. The data blocks that the guest OS does not use are not stored. To implement this, the two-level structure is used. The VM (VEE) knows which blocks

have data. Linux does not have such feature. The claimed combination operates below the level of Linux and a file system. Therefore, Warren, even used in different interpretation, will not produce the claimed combination.

Regarding claim 5, Warren teaches that Linux has hierarchy display of the directory system. However, it is different from claimed redirection blocks that “have multilevel hierarchy.” Linux does not have redirection blocks. Therefore, claim 5 is not obvious over Warren for this additional reason.

Claim 17 recites that “each VM uses a common storage device driver and a common file system driver.” Warren teaches creating a single VM on each OS. The VMs of Warren use shared disk (see page 316). However, this is different from multiple VMs running on the same OS kernel, where each VM shares a common storage device driver and a common file system driver. Therefore, claim 17 is not obvious over Warren for this additional reason.

Claims 4, 9 and 10 stand rejected as being allegedly unpatentable over Warren in view of Karpoff. Claim 4 recites “at least one file has a B-tree or B+tree structure.” The Office Action relies on Karpoff for this feature. Karpoff teaches a system for storage virtualization (see abstract). Karpoff discusses B-Tree structure of the files. However, neither Warren nor Karpoff teach the file as claimed. Claim 4, as amended, depends from claim 1 that recites “the file of the virtual drive comprises **an internal structure** that includes user data storage blocks and redirection blocks that point to user data storage blocks.” Neither Warren nor Karpoff teach a file having an internal structure as claimed. Therefore, even if the file of Warren were to be combined with B-Tree structure of Karpoff, it would not produce the claimed combination.

Claim 9 recites “the disk image files are reduced in size to exclude data corresponding to unused blocks of the virtual disk drive.” The Office Action relies on Karpoff (column 9, lines 64-67) for the claimed feature. However, the image files of Karpoff are not the files having an internal structure, as claimed. Therefore, Warren combined with Karpoff will not produce the claimed combination. Claim 10 is rejected based on the same reasons as claim 9. Therefore, claim 10 is not obvious over Warren and Karpoff for the same reason as claim 9.

Claim 19 stands rejected as being allegedly unpatentable over Warren in view of Goldsmith. Claim 19, as amended, recites that “the disk image comprises an internal structure that includes user data storage blocks and redirection blocks that point to user data storage blocks and that are used by the VM-specific file system driver.” Neither Warren nor Goldsmith teaches this feature. Therefore, combination of Warren with Goldsmith will not produce the claimed combination.

Claim 20 stands rejected as being allegedly unpatentable over Warren in view of Kim. Claim 20, as amended, recites that “the internal structure includes user data storage blocks and redirection blocks that point to user data storage blocks and that are used by the VM-specific file system driver.” Neither Warren, nor Kim teaches this feature. Therefore, combination of Warren with Kim will not produce the claimed combination.

Conclusion

All of the stated grounds of objection and rejection have been properly traversed, accommodated, or rendered moot. Applicants therefore respectfully request that the Examiner reconsider all presently outstanding objections and rejections and that they be withdrawn.

- 16 -

Protassov *et al*
Appl. No. 11/757,598

Applicants believe that a full and complete reply has been made to the outstanding Office Action and, as such, the present application is in condition for allowance. If the Examiner believes, for any reason, that personal communication will expedite prosecution of this application, the Examiner is invited to telephone the undersigned at the number provided.

Prompt and favorable consideration of this Amendment and Reply is respectfully requested.

Respectfully submitted,

BARDMESSER LAW GROUP

/GB/

George S. Bardmesser
Attorney for Applicant
Registration No. 44,020

Date: August 3, 2011

1025 Connecticut Avenue, N.W., Suite 1000
Washington, D.C. 20006
(202) 293-1191

Electronic Acknowledgement Receipt

EFS ID:	10655048
Application Number:	11757598
International Application Number:	
Confirmation Number:	2963
Title of Invention:	SYSTEM AND METHOD FOR MANAGEMENT OF VIRTUAL EXECUTION ENVIRONMENT DISK STORAGE
First Named Inventor/Applicant Name:	STANISLAV S. PROTASSOV
Customer Number:	54089
Filer:	George Simon Bardmesser
Filer Authorized By:	
Attorney Docket Number:	2230.0850001
Receipt Date:	03-AUG-2011
Filing Date:	04-JUN-2007
Time Stamp:	11:01:48
Application Type:	Utility under 35 USC 111(a)

Payment information:

Submitted with Payment	no
------------------------	----

File Listing:

Document Number	Document Description	File Name	File Size(Bytes)/ Message Digest	Multi Part /.zip	Pages (if appl.)
1		AMENDMENTAFTERFIRSTOFFICE ACTION.pdf	81873 86aac097231fc63b7e172790dd139924fccd1ae3	yes	16

Multipart Description/PDF files in .zip description

	Multipart Description/PDF files in .zip description		
	Document Description	Start	End
	Amendment/Req. Reconsideration-After Non-Final Reject	1	2
	Claims	3	8
	Applicant Arguments/Remarks Made in an Amendment	9	16

Warnings:**Information:****Total Files Size (in bytes):**

81873

This Acknowledgement Receipt evidences receipt on the noted date by the USPTO of the indicated documents, characterized by the applicant, and including page counts, where applicable. It serves as evidence of receipt similar to a Post Card, as described in MPEP 503.

New Applications Under 35 U.S.C. 111

If a new application is being filed and the application includes the necessary components for a filing date (see 37 CFR 1.53(b)-(d) and MPEP 506), a Filing Receipt (37 CFR 1.54) will be issued in due course and the date shown on this Acknowledgement Receipt will establish the filing date of the application.

National Stage of an International Application under 35 U.S.C. 371

If a timely submission to enter the national stage of an international application is compliant with the conditions of 35 U.S.C. 371 and other applicable requirements a Form PCT/DO/EO/903 indicating acceptance of the application as a national stage submission under 35 U.S.C. 371 will be issued in addition to the Filing Receipt, in due course.

New International Application Filed with the USPTO as a Receiving Office

If a new international application is being filed and the international application includes the necessary components for an international filing date (see PCT Article 11 and MPEP 1810), a Notification of the International Application Number and of the International Filing Date (Form PCT/RO/105) will be issued in due course, subject to prescriptions concerning national security, and the date shown on this Acknowledgement Receipt will establish the international filing date of the application.

PTO/SB/06 (07-06)

Approved for use through 1/31/2007. OMB 0651-0032
U.S. Patent and Trademark Office; U.S. DEPARTMENT OF COMMERCE

Under the Paperwork Reduction Act of 1995, no persons are required to respond to a collection of information unless it displays a valid OMB control number.

PATENT APPLICATION FEE DETERMINATION RECORD Substitute for Form PTO-875					Application or Docket Number 11/757,598		Filing Date 06/04/2007		<input type="checkbox"/> To be Mailed	
APPLICATION AS FILED – PART I										
(Column 1)			(Column 2)			SMALL ENTITY <input checked="" type="checkbox"/> OR		OTHER THAN SMALL ENTITY		
FOR	NUMBER FILED	NUMBER EXTRA	RATE (\$)	FEE (\$)	OR	RATE (\$)	FEE (\$)			
<input type="checkbox"/> BASIC FEE (37 CFR 1.16(a), (b), or (c))	N/A	N/A	N/A			N/A				
<input type="checkbox"/> SEARCH FEE (37 CFR 1.16(k), (l), or (m))	N/A	N/A	N/A			N/A				
<input type="checkbox"/> EXAMINATION FEE (37 CFR 1.16(o), (p), or (q))	N/A	N/A	N/A			N/A				
TOTAL CLAIMS (37 CFR 1.16(j))	minus 20 =	*	X \$	=		X \$	=			
INDEPENDENT CLAIMS (37 CFR 1.16(h))	minus 3 =	*	X \$	=		X \$	=			
<input type="checkbox"/> APPLICATION SIZE FEE (37 CFR 1.16(s))	If the specification and drawings exceed 100 sheets of paper, the application size fee due is \$250 (\$125 for small entity) for each additional 50 sheets or fraction thereof. See 35 U.S.C. 41(a)(1)(G) and 37 CFR 1.16(s).									
<input type="checkbox"/> MULTIPLE DEPENDENT CLAIM PRESENT (37 CFR 1.16(j))										
			TOTAL			TOTAL				
* If the difference in column 1 is less than zero, enter "0" in column 2.										
APPLICATION AS AMENDED – PART II										
(Column 1)			(Column 2)			SMALL ENTITY OR		OTHER THAN SMALL ENTITY		
AMENDMENT	08/03/2011	CLAIMS REMAINING AFTER AMENDMENT	HIGHEST NUMBER PREVIOUSLY PAID FOR	PRESENT EXTRA	RATE (\$)	ADDITIONAL FEE (\$)	OR	RATE (\$)	ADDITIONAL FEE (\$)	
	Total (37 CFR 1.16(i))	* 22	Minus	** 22	= 0	X \$26 =	0	OR	X \$	=
	Independent (37 CFR 1.16(h))	* 7	Minus	*** 7	= 0	X \$110 =	0	OR	X \$	=
	<input type="checkbox"/> Application Size Fee (37 CFR 1.16(s))							OR		
	<input type="checkbox"/> FIRST PRESENTATION OF MULTIPLE DEPENDENT CLAIM (37 CFR 1.16(j))							OR		
						TOTAL ADD'L FEE	0	OR	TOTAL ADD'L FEE	
(Column 1)			(Column 2)			SMALL ENTITY OR		OTHER THAN SMALL ENTITY		
AMENDMENT	CLAIMS REMAINING AFTER AMENDMENT	HIGHEST NUMBER PREVIOUSLY PAID FOR	PRESENT EXTRA	RATE (\$)	ADDITIONAL FEE (\$)	OR	RATE (\$)	ADDITIONAL FEE (\$)		
	Total (37 CFR 1.16(i))	*	Minus	**	=	X \$	=	OR	X \$	=
	Independent (37 CFR 1.16(h))	*	Minus	***	=	X \$	=	OR	X \$	=
	<input type="checkbox"/> Application Size Fee (37 CFR 1.16(s))							OR		
	<input type="checkbox"/> FIRST PRESENTATION OF MULTIPLE DEPENDENT CLAIM (37 CFR 1.16(j))							OR		
						TOTAL ADD'L FEE		OR	TOTAL ADD'L FEE	
<p>* If the entry in column 1 is less than the entry in column 2, write "0" in column 3.</p> <p>** If the "Highest Number Previously Paid For" IN THIS SPACE is less than 20, enter "20".</p> <p>*** If the "Highest Number Previously Paid For" IN THIS SPACE is less than 3, enter "3".</p> <p>The "Highest Number Previously Paid For" (Total or Independent) is the highest number found in the appropriate box in column 1.</p>										

Legal Instrument Examiner:
/ROSALIND BALL/

This collection of information is required by 37 CFR 1.16. The information is required to obtain or retain a benefit by the public which is to file (and by the USPTO to process) an application. Confidentiality is governed by 35 U.S.C. 122 and 37 CFR 1.14. This collection is estimated to take 12 minutes to complete, including gathering, preparing, and submitting the completed application form to the USPTO. Time will vary depending upon the individual case. Any comments on the amount of time you require to complete this form and/or suggestions for reducing this burden, should be sent to the Chief Information Officer, U.S. Patent and Trademark Office, U.S. Department of Commerce, P.O. Box 1450, Alexandria, VA 22313-1450. DO NOT SEND FEES OR COMPLETED FORMS TO THIS ADDRESS. **SEND TO: Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450.**

If you need assistance in completing the form, call 1-800-PTO-9199 and select option 2.